ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PROJET D'APPLICATION PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE LOGICIEL
M. ING.

PAR
WAJDI ABBOUD

MODÉLISATION ET CONCEPTION D'UNE BASE DE DONNÉES ORIENTÉE-OBJET POUR L'INSPECTION AUTOMATISÉE DE PIÈCES MANUFACTURÉES

MONTRÉAL, AVRIL 1999
© Tous droits réservés, Wajdi Abboud 1999

CE PROJET D'APPLICATION A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

- M. Richard Lepage, professeur-tuteur et professeur
  au Département de génie de la production automatisée à l'École de technologie supérieure

- M. Michel Lavoie, professeur
  au Département de génie électrique à l'École de technologie supérieure

- M. Tanneguy Redarce,
  Maître de conférences à l'Institut national des sciences appliquées de Lyon

IL A FAIT L'OBJET D'UNE PRÉSENTATION DEVANT CE JURY ET UN PUBLIC

LE 24 MARS 1999

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

MODÉLISATION ET CONCEPTION D'UNE BASE DE DONNÉES ORIENTÉE-OBJET POUR L'INSPECTION AUTOMATISÉE DE PIÈCES MANUFACTURÉES

Wajdi Abboud

(Sommaire)

Dans le cadre d'un projet de collaboration entre la France et le Québec intitulé: ″*Inspection automatisée de pièces manufacturées*″, ce travail consiste à développer une base de données centralisée en langage ″JAVA″ pouvant contenir toutes les informations relatives à des pièces manufacturées et à développer des outils informatisés permettant d'interroger la BD à distance.

Géographiquement, les utilisateurs de cette BD sont répartis entre plusieurs laboratoires français et canadiens (LAI, LIVIA, CNRC, …).

Pour interroger à distance le système, deux outils ont été développés dans ce travail: un pour la partie serveur et l'autre pour la partie client. La partie serveur contient la BD avec toutes les informations stockées sous forme de fichiers binaires et l'autre est constituée d'un programme permettant de questionner le serveur à distance dans le but de visualiser ou de compléter la BD et ceci grâce à une interface graphique à usage simple.

La méthodologie appliquée à la réalisation de ce projet s'appuie sur:

- L'analyse orientée objet (AOO) qui permet de clarifier les divers problèmes rencontrés au cours de notre projet.
- Le langage de programmation ″JAVA″ avec l'environnement de développement JDK de SUN. Ainsi pour optimiser la BD d'une façon efficace, nous avons pensé utiliser des fichiers en format binaire. En effet, les accès se feront ainsi plus rapidement.
- La douille Java, utilisant le protocole TCP/IP comme langage de communication entre un client et le serveur de la BD.

Avec la douille Java, se basant sur le protocole TCP/IP, nous avons assuré un haut niveau d'interaction. Ce dernier est caractérisé par un court temps de réponse surtout quand par exemple, un utilisateur autorisé veut télécharger chez lui des icônes de pièces manufacturées ainsi que des fichiers reliés à ces dernières. Nous avons atteint des résultats satisfaisants grâce au téléchargement rapide et à la facilité d'accès à la BD.

Enfin, nous suggérons des améliorations pour augmenter la performance de ce système. Une technologie prometteuse est le concept des Agents Mobiles (AM). Ce dernier aura le rôle de lancer une requête à la BD. En revanche, cette innovation technologique est très jeune. Nous avons évité de l'utiliser comme stratégie d'implantation dans notre projet, en raison que son usage est très précoce pour le moment et elle est au stade de développement et de validation.

DESIGN AND MODELING OF OBJECT-ORIENTED DATABASE FOR AUTOMATED INSPECTION OF MANUFACTURED COMPONENTS

Wajdi Abboud

(Abstract)

This work is the result of a France-Canada joint project called *"Automated Inspection of Manufactured Components"*. In this project, we defined a centralized database containing all the necessary information on different manufactured parts. We also developed computer tools needed for remote database queries.

The users of this database are located in different laboratories in France and in Canada.

In order to accomplish the so-called remote database access, we created two related tools for database interfacing: a server module and a client module using client-server programming paradigm. The goal is to simplify processing and to present a uniform model for data retrieval.

The framework used in the design and modelling consists of:

- Object Oriented Analysis (OOA) which enables the development of high-quality tools by defining the problem structure.
- The JAVA language which provides a robust development environment. Furthermore, by using binary encoding schemes, data access time can also be reduced.
- The Java's socket implementation with TCP/IP protocol. These JAVA sockets permits a high-level abstraction for our client-server modules.

By using the Java sockets, we were able to obtain a fast system response time. The fast response time is especially useful in the remote loading of icon objects and related data.

Further improvements may also be achieved with the new *Mobile Agent Concept*. A Mobile Agent can query the database without direct handling by neither client-side module nor server-side module. Thus, the required bandwidth for normal database transactions is reduced.

However, the *Mobile Agent Concept* is still an emerging technology. That's why further investigation should be carried out to determine the Mobile Agent Concept's capabilities and limitations.

REMERCIEMENTS

La question à se poser finalement est de savoir si la vie est une expérience à vivre ou un problème à résoudre[1]

Je tiens à remercier toutes les personnes qui, de près ou de loin, m'ont soutenu dans la réalisation de ce travail. Particulièrement,

Je voudrais exprimer ma profonde gratitude envers Dr Richard Lepage, Professeur au département du génie de la production automatisée (GPA) à l'École de technologie supérieure à Montréal, qui a dirigé ce travail. J'apprécie sincèrement, non seulement ses compétences scientifiques, mais aussi la patience, la compréhension et l'aide efficace qu'il a témoigné à mon égard et qui m'ont permis de finaliser ce travail.

Je remercie Messieurs les Professeurs Michel Lavoie et Tanneguy Redarce de leur engagement en tant que membres du jury d'évaluation de ce travail.

Je n'oublie pas de remercier sincèrement Matthieu Aubry, stagiaire de l'INSA-Lyon, qui a contribué à l'implantation et a favorisé le développement des idées contenues dans ce travail.

Enfin, du fond de mon cœur, je remercie tous les membres de ma famille. Leur amour, leur abnégation, leurs encouragements ont été mes motivations tout au long de mes études.

[1] Pelletier, Denis, L'arc-en-soi, Robert Laffont/Stanké, Montréal 1981.

TABLE DES MATIÈRES

SOMMAIRE i
ABSTRACT iii
REMERCIEMENTS v
TABLE DES MATIÈRES vi
LISTE DES TABLEAUX x
LISTE DES FIGURES xi
LISTE DES ABRÉVIATIONS ET DES SIGLES xii

INTRODUCTION 1

CHAPITRE 1 :
DESCRIPTION DU PROBLÈME 3

1.1 Définition du projet d'inspection automatisée 3
1.1.1 Contraintes du projet 4
1.1.2 Définition du problème 6
1.1.3 Objectifs 8
1.1.4 Originalité 8
1.2 Théorie des géons 9
1.2.1 Reconnaissance par composantes 9
1.2.2 Description par géons 12
1.3 Attributs intrinsèques 14
1.3.1 Attributs des géons 14

1.3.2 Attribut de connectivité 15
1.4 Répertoire de géons 16
1.5 Pièces de la base de données 17
1.6 Description géon des pièces 18
1.6.1 Géons constitutifs 18
1.6.2 Inventaire des géons utilisés 18
1.6.3 Description complète avec connectivité 19
1.7 Base de données 20
1.7.1 Données associées à chaque pièce 20

CHAPITRE 2 :
MODÉLISATION ORIENTÉE-OBJET DE LA BASE DE DONNÉES 22

2.1 Pourquoi avons-nous besoin de l'analyse orientée-objet? 22
2.2 Définitions 26
2.2.1 Objet 26
2.2.2 Classe et instance 26
2.2.3 Attribut 27
2.2.4 Méthode 28
2.2.5 Constructeur 28
2.2.6 Connexion d'instance 28
2.2.7 Message et polymorphisme 29
2.2.8 Structure Composé-Composants 30
2.2.9 Structures Généralisation-Spécialisations 30
2.2.9.1 Notion d'héritage 31
2.2.10 Modélisation 31
2.3 Langages de programmation et environnement de développement 32
2.3.1 Pourquoi JAVA? 32
2.3.2 Environnement de développement adapté à notre plate-forme 33

2.4 Introduction à la base de données 35
2.4.1 Avantages et inconvénients 35
2.4.2 Vue globale du système d'inspection automatisée 36
2.4.3 Besoins pour la base de données d'inspection 37
2.4.4 Buts principaux associés à la base de données 37
2.5 Système de gestion de base de données (SGBD) 38
2.5.1 Concepts généraux des SGBD 38
2.5.2 Principaux modèles sur lesquels s'appuient les SGBD 39
2.6 Les systèmes de gestion de fichiers (SGF) 40
2.6.1 Définition d'un fichier 40
2.6.2 Les fichiers de la base de données 40
2.6.3 Objectifs des SGF 41
2.6.4 Organisation de fichiers 41
2.7 Modèle conceptuel de données (MCD) 42
2.8 Modélisation Orientée-Objet 44
2.9 Associations entre les classes de la base 46

CHAPITRE 3 :
ARCHITECTURE CLIENT-SERVEUR DE LA BASE DE DONNÉES 49

3.1 Les douilles d'UNIX 49
3.1.1 Définition 49
3.1.2 Introduction 50
3.1.3 Les primitives TCP pour les douilles 51
3.1.4 Enchaînement des procédures 51
3.1.4.1 Mode connecté 52
3.2 Le protocole TCP/IP 54
3.2.1 TCP/IP et JAVA 55
3.2.2 Principe d'utilisation des douilles 55

3.2.2.1 Coté Client..... 56
3.2.2.2 Coté Serveur..... 56
3.3 Architecture Client-Serveur 57
3.3.1 La répartition Client-Serveur 58
3.4 Liste des requêtes acceptées par le serveur..... 59
3.5 Les fenêtres utilisées pour la base de données..... 60

CHAPITRE 4 :
CONCLUSION ET RECOMMANDATION..... 61

BIBLIOGRAPHIE 64

ANNEXE A : DIALOGUE CLIENT-SERVEUR 67
ANNEXE B : INVENTAIRE DES FLUX ET MATRICE DES FLUX
DE DONNÉES 69
B.1 Inventaire des flux 70
B.2 Matrice des flux des données..... 71
ANNEXE C : MANUEL D'UTILISATION 73
C.1 Programmes 74
C.2 Accès aux programmes 75
ANNEXE D : LE DICTIONNAIRE DES CLASSES..... 76
ANNEXE E : LISTE DES REQUÊTES ACCEPTÉES PAR LE SERVEUR 104
ANNEXE F : LES FENÊTRES UTILISÉES POUR LA BASE DE DONNÉES 111

LISTE DES TABLEAUX

Tableau Page

1.1 Valeurs d'attributs intrinsèques 15
1.2 Valeurs de l'attribut de connectivité 15
1.3 Répertoire des géons 16

LISTE DES FIGURES

Figure		Page
1.1	Inspection automatisée de pièces manufacturées	5
1.2	Attributs du géon	11
1.3	Graphe de géons	12
1.4	Géons semblables	13
1.5	Pièces de la base de données	17
1.6	Description complète avec connectivité des pièces	19
1.7	Vue à deux dimensions d'une pièce de la base	21
2.1	Représentations graphiques de l'AOO	25
2.2	Notation pour l'instantiation	26
2.3	Les messages	29
2.4	Application partageant la base d'inspection	36
2.5	Organisation de fichiers de la base de données	41
2.6	Le modèle conceptuelle de données (MCD)	43
2.7	Le modèle Orienté-Objet	44
3.1	Enchaînement des primitives pour le mode connecté	52
3.2	Scénario entre Client et Serveur	56
3.3	Dialogue Client-Serveur	57
3.4	Répartition Client-Serveur	58
3.5	Client-Serveur et Canal	59

LISTE DES ABRÉVIATIONS ET DES SIGLES

AM	:	Agents Mobiles
AOO	:	Analyse Orientée-Objet
API	:	*Application Program Interface*
BD	:	Base de données
CAO	:	Conception Assistée par Ordinateur
CG	:	Centre de gravité
CNRC	:	Conseil national de recherches du Canada
CRIM	:	Centre de recherche informatique de Montréal
ÉTS	:	École de technologie supérieure
FAO	:	Fabrication assistée par ordinateur
INSA	:	Institut national des sciences appliquées (France)
IP	:	*Internet Protocol*
JDK	:	*Java Developper Kit*
LAI	:	Laboratoire d'automatique industrielle
LDD	:	Langage de définition de données
LIVIA	:	Laboratoire d'Imagerie, de Vision et d'Intelligence Artificielle
LMD	:	Langage de manipulation de données
MCD	:	Modèle conceptuel des données
OO	:	Orienté-Objet
POO	:	Programmation Orientée-Objet
SGBD	:	Système de gestion de base de données
TCP	:	*Transmission Control Protocol*
2D	:	Deux dimensions
3D	:	Tridimensionnel (trois dimensions)

INTRODUCTION

De nos jours, l'utilité des bases de données (BD) n'est plus à démontrer. En effet, pour un projet de grande envergure, il est souvent nécessaire de stocker des données et d'y accéder le plus rapidement possible. Ainsi, ce projet d'application consiste à structurer une base de données centralisée pouvant contenir toutes les informations relatives à des pièces mécaniques et à développer des outils informatisés permettant d'interroger la base de données localement et à distance: Ceci rentre dans le cadre d'un projet en collaboration entre la France et le Québec intitulé: *″Inspection Automatisée de Pièces Manufacturées″*.

Le développement de deux outils (un pour la partie serveur et un pour la partie client) était donc nécessaire afin de pouvoir interroger à distance le système. La partie serveur contient la BD proprement dite avec toutes les informations stockées sous forme de fichiers binaires. La partie client est constituée d'un programme permettant de questionner le serveur à distance dans le but de visualiser ou de compléter la BD et ceci grâce à une interface graphique simple et conviviale.

Géographiquement, les utilisateurs sont répartis entre plusieurs laboratoires français et canadiens (LAI, LIVIA, CNRC, …). Ils désirent accéder à la BD qui est fonction des besoins de chacun d'eux. Par exemple, le système permet à un utilisateur du laboratoire LAI à Lyon d'accéder et d'échantillonner quelques pièces de la BD laquelle est localisée physiquement sur le serveur de l'école de technologie supérieure de Montréal.

Pour garder un meilleur contrôle sur la BD, un système d'exploitation UNIX restreint et autorise l'accès à un groupe d'usagers privilégiés déterminés au préalable.

La réalisation de ce projet s'appuie sur la méthodologie suivante:

- L'analyse utilisée dans ce rapport d'application est Orientée-Objet (OO). Elle est inspirée de plusieurs auteurs comme Coad et Yourdon, Davis, Rumbaugh et autres [Coad et Yourdon, 1992; Rumbaugh et al., 1996; Davis, 1993]. L'utilité de cette analyse est de spécifier la BD. Ainsi, nous avons utilisé la méthode française ″Merise″ [Gabay, 1993] pour spécifier le modèle conceptuel de données (MCD) ainsi que la matrice des flux de données. Ces derniers apparaissent en chapitre 2 et en annexe B à la fin de ce document.
- Pour réaliser le développement de notre projet, nous avons utilisé ″JAVA″ comme langage de programmation avec l'environnement de développement *JDK* (*Java Developer Kit*).
- Une douille Java, utilisant le protocole TCP/IP comme langage de communication, est implantée permettant d'assurer la communication avec le serveur de la BD d'un client distant.

Ce mémoire se divise en trois chapitres et une conclusion:

- Le premier expose la problématique, il décrit le projet d'inspection automatisée et définit le problème associé à la BD
- Le deuxième illustre la méthode d'analyse proposée pour le projet ainsi que le cadre conceptuel que nous avons adopté pour la réalisation. Il se termine avec une modélisation Orientée-Objet de la BD
- Le troisième chapitre approfondit les concepts reliés à la communication Client-Serveur à partir de la douille et du protocole TCP/IP

Enfin, une conclusion clôt le présent document.

CHAPITRE 1

DESCRIPTION DU PROBLÈME

Pour permettre une bonne compréhension du travail présenté dans ce document, il est opportun de clarifier le projet et de rappeler certains concepts liés au sujet. Ce chapitre présente une description théorique détaillée des géons, ainsi que les principales contraintes qui sont reliées à la construction de la base de données.

1.1 Définition du projet d'inspection automatisée

L'inspection automatisée d'objets manufacturés consiste à comparer leur modèle de conception[1] avec leur image 3D obtenue à l'aide d'un capteur télémétrique laser. La mise en registre de ces deux représentations permet de conclure si une pièce réalisée est conforme à son modèle théorique défini lors de sa conception. La mise en correspondance est un problème difficile, complexifié par la nécessité de sélectionner le bon modèle dans une base de données CAO de grande taille qui contient la description de toutes les pièces susceptibles de se retrouver sur la chaîne d'inspection.

Afin d'améliorer sensiblement le temps de réponse pour la sélection du bon modèle dans la base de données, les partenaires du projet ont proposé d'établir des

---

[1] De nombreux outils informatiques facilitent la conception des pièces (CAO: conception assistée par ordinateur) de même que, de plus en plus, leur fabrication (FAO: fabrication assistée par ordinateur). Ce foisonnement d'outils informatiques a permis de diminuer considérablement les temps de conception et de fabrication de pièces en série.

modèles visuels basés sur des géons, tant des modèles CAO que des modèles de l'objet sous inspection. Les modèles visuels 2D obtenus sont moins précis mais beaucoup plus compacts, ce qui permettra au départ une mise en correspondance rapide des modèles visuels, permettant ensuite de sélectionner le modèle CAO le plus approprié pour le module d'inspection.

Ce projet se déroule dans le cadre d'une coopération entre la France et le Québec et vise d'arriver à un système tel qu'il est représenté sur le schéma suivant (Figure 1.1). Le présent document se concentre sur la conception et le développement de la base de données qui centralise les échanges de données dans le projet d'inspection automatisée.

1.1.1 Contraintes du projet

Un géon est un volume élémentaire descriptible par cylindre généralisé (CG). Une pièce manufacturée quelconque est constitué d'au moins 1 géon. Chaque géon est caractérisé par cinq attributs intrinsèques explicités plus loin. Un attribut de connectivité précise le type de relation spatiale entre les géons interreliés. Un géon peut n'avoir aucun voisin, un seul voisin, ou plusieurs voisins.

Actuellement, la base de données contient 10 pièces. Chacune de ces pièces possède obligatoirement deux fichiers, un pour la description CAO et l'autre pour l'aspect visuel .GIF. Une pièce peut avoir aucun ou plusieurs fichiers 2D, comme elle peut être membre de plusieurs bases (bases réduites).

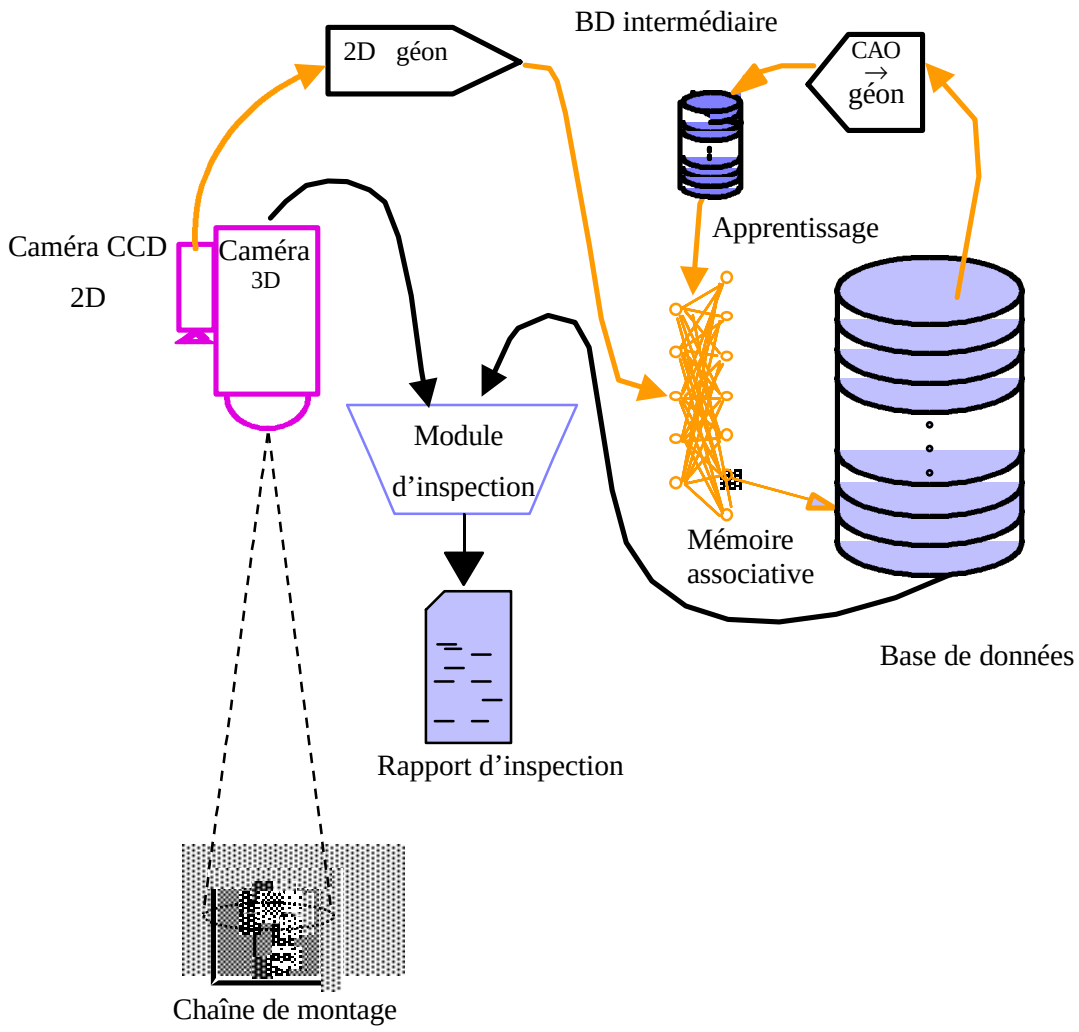


Figure 1.1: Inspection automatisée de pièces manufacturées

La pièce sur la chaîne de montage est inspectée par une caméra tridimensionnelle dont la sortie alimente le module d'inspection et par une caméra CCD dont l'image de sortie, décomposée en ses géons constituants, alimente par la suite la mémoire associative qui permet d'obtenir rapidement le modèle approprié dans la base de données. La représentation surfacique du modèle sélectionné alimente également le module d'inspection qui calcule la meilleure mise en registre et permet donc de détecter les défauts de fabrication et les écarts de tolérance.

1.1.2 Définition du problème

> *On peut comparer un projet informatique à un casse-tête que nous essayons de résoudre. Avant de commencer, il faut déterminer le problème que l'on veut solutionner. En d'autres termes, nous devons d'abord définir le problème* [Hurtubise, 1984] .

La définition du problème pour notre application informatique a commencé avec une vague description comme: mettre en œuvre une structuration Orientée-Objet (en Java) pour une base de données (BD) partagée entre plusieurs laboratoires et développer un outil d'interrogation pour questionner la BD à distance et localement. Toutefois, nous nous devons d'être plus spécifiques si nous voulons que le développement du système se fasse efficacement.

Qui seront les principaux utilisateurs du système? Sont-ils familiers à utiliser une BD? … L'expérience des utilisateurs sera donc un élément clé dans le développement d'un système de gestion de base de données (SGBD).

Cette base de données constitue un besoin crucial pour la réalisation du projet d'inspection. Par exemple, un utilisateur autorisé a besoin d'accéder à cette base pour télécharger chez lui des fichiers reliés à des pièces mécaniques quelconques. D'autres ont besoin d'accéder à cette dernière pour ajouter des nouvelles pièces manufacturées et pour joindre des fichiers à ces pièces.

Les utilisateurs de la base de données sont distribués entre plusieurs laboratoires. Pour valider les différents accès par ces utilisateurs privilégiés, on a jugé nécessaire de les gérer par un SGBD détaillé au chapitre 2.

Notre base de données est centralisée, ses données sont stockées sur le serveur de l'ÉTS sous forme de fichiers binaires. Elle doit être accessible avec un temps de réponse[2] rapide ou minimal. Pour ce faire, on a pensé stocker les données de la base dans des fichiers binaires sous forme de table d'adressage calculé (*Hashtable*) bien indexée, ce qui rend le temps de réponse très rapide.

Pour qu'un client communique avec le serveur de l'ÉTS, on a besoin d'un protocole de communication (Client-Serveur). La douille TCP/IP (*Transmission Control Protocol/Internet Protocol*) répond à ce besoin puisqu'elle permet par exemple de télécharger dans un temps minimal les icônes .GIF du serveur vers le compte de l'utilisateur. Ainsi à partir de la douille, un client peut questionner le serveur à distance et en local. Cet outil est décrit au chapitre 3.

Enfin, pour mettre en valeur la base de données, nous avons développé un outil informatisé en java permettant de répondre aux caractéristiques désirées de la BD. Cet outil est décrit à l'annexe F.

La mise en œuvre et le développement cités précédemment sont les aspects majeurs de ce travail de recherche.

---

[2] Le temps écoulé entre une demande de traitement et son exécution complétée.

1.1.3 Objectifs

L'objectif général du projet de recherche est d'automatiser l'inspection de pièces manufacturées disposées sur une chaîne de montage/inspection. La réalisation du projet nécessite:

- la structuration d'une base de données qui contient les fichiers produits lors de la conception des diverses pièces et le développement d'un outil d'interrogation pour questionner la base de données à distance et en local. Justement, le but de mon projet s'appuie sur ce point précis.

- le développement de mécanismes robustes de mise en correspondance des images 3D provenant du télémètre et des données décrivant le modèle CAO sélectionné de la base de données, et enfin

- le développement d'un système de sélection rapide du modèle approprié CAO basé sur une représentation visuelle, moins précise mais plus compacte, de la pièce sous inspection et des modèles CAO contenus dans la base de données.

1.1.4 Originalité

Avec ce projet, nous cherchons à introduire des principes de vision artificielle afin de simplifier les modes de représentation qui sont utilisés dans l'environnement de la conception assistée par ordinateur, et ainsi accélérer considérablement certains calculs (comme le choix du bon modèle dans une base de données) qui ne nécessitent pas la grande précision requise ailleurs.

1.2 Théorie des géons

La théorie des géons de Biederman est une théorie de la perception visuelle humaine. Cette dernière décrit les objets qui sont complexes sous forme de composants simples: les géons (cylindre, cube, sphère, …). Les informations concernant une pièce manufacturée sont stockées dans notre BD sous forme de géons.
Dans notre projet, nous essayons d'appliquer cette théorie pour prouver son fonctionnement. Dans cette optique, nous nous servirons des géons pour décrire des pièces manufacturées afin d'accélérer l'indexation de celles-ci dans la BD. En effet, la décomposition d'une pièce sous forme de géons offre une représentation simple de celle-ci, qui permet de restreindre la quantité d'information concernant l'indexage.

1.2.1 Reconnaissance par composantes

Dans la foulée du principe de *Recognition by parts* proposé par [Binford, 1971], Biederman (1985-1987) formulait dans les années 80 sa théorie de *Reconnaissance par composantes* (RBC, pour *Recognition By Components*), par laquelle il tentait d'expliquer comment une scène complexe est analysée à prime abord par le système visuel humain. Tel que son nom le laisse entendre, la reconnaissance par composantes présume que les objets présents dans le champ de vision sont découpés mentalement selon leurs parties naturelles, nommément les *géons;* cet assemblage grossier de volumes élémentaires servirait de clé pour l'identification sommaire de l'objet. La simplicité du concept a incité plusieurs chercheurs en vision à reprendre cette théorie dans un contexte de reconnaissance d'objets 3D dans une image [Dickinson et al., 1992].

Fondamentalement, un géon pourrait avoir n'importe quelle forme 3D; cependant, Biederman restreint les géons à la famille des cylindres généralisés (CG) afin de rendre sa théorie plus plausible. Ce choix a d'ailleurs été maintenu dans la majorité des applications de vision artificielle, puisque les CGs permettent de modéliser bon

nombre d'objets courants, naturels ou manufacturés, tout en limitant le degré de difficulté inhérent au problème.

Malgré les diverses représentations possibles d'un CG, Biederman retient seulement quatre attributs symboliques pour caractériser un géon, dont les différentes possibilités sont illustrées à la figure 1.2:

- Axe : droit ou recourbé;
- Section:
  - arêtes : droites ou recourbées;
  - symétrie : réflective, rotationnelle et réflective, ou asymétrique;
  - dimension : constante, en expansion, ou en expansion et contraction.

Ce choix est d'ailleurs conséquent avec le principe d'une reconnaissance de prime abord (*primal access*), qui exclut toute métrique complexe dans l'opération mentale de découpage. Il reste qu'un objet réel est généralement constitué de plusieurs géons: aussi un attribut de connectivité doit-il être spécifié pour chaque paire de géons, de sorte à traduire la position relative des géons. Dans ses travaux, Biederman postule trois types de relations spatiales, certaines d'ordre géométrique, et d'autres d'ordre perceptuel:

- Type de connexion: bout-à-bout ou par le côté;
- Point d'attache: à une surface courte ou longue;
- Position relative: au-dessus ou en-dessous.

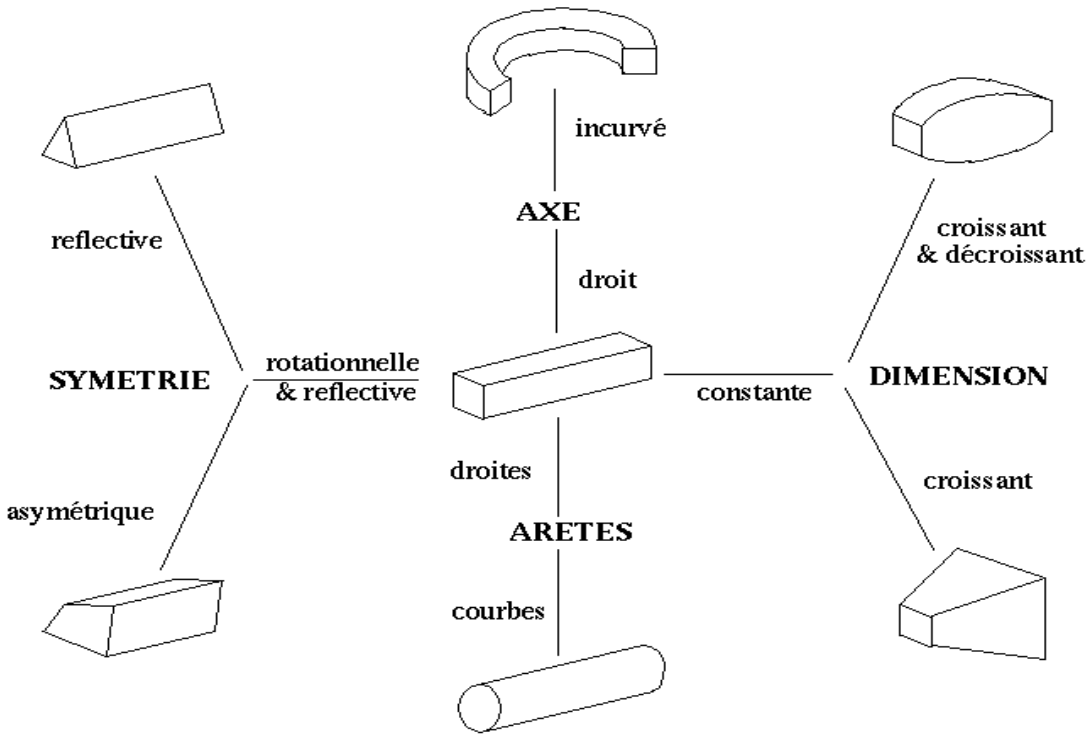


Figure 1.2: Attributs du géon

Les attributs utilisés pour décrire chaque géon sont résumés au tableau 1.1

Pour répondre à nos besoins spécifiques ainsi que pour l'adapter plus à notre problématique en inspection des pièces manufacturées, nous avons rajouté à la théorie de Biederman un nouvel attribut nommé *″Matérialité″* qui peut prendre deux valeurs différentes: +1 signifie un géon solide qui ne possède pas de trous et –1 signifie un géon cavité pour modéliser les trous. Aussi, nous avons fusionné deux valeurs concernant l'attribut *″Dimension″* de la figure 1.2 en une seule catégorie qui s'appelle variable (en plus de l'attribut *″constante″*). De même, nous avons ajouté à l'attribut *″Arêtes″* la valeur hybride quand un profit générateur possède des arêtes droites et des arêtes courbes en même temps.

1.2.2 Description par géons

La représentation par géons est une description structurelle des objets, généralement codifiée sous forme de graphe relationnel. À chaque géon est associé un nœud du graphe, caractérisé par les attributs intrinsèques du géon; les arcs, dirigés ou non, expriment les relations d'adjacence entre les différents géons [Grégoire, 1996]. La figure 1.3 illustre un exemple de graphe relationnel pour une des pièces du projet.

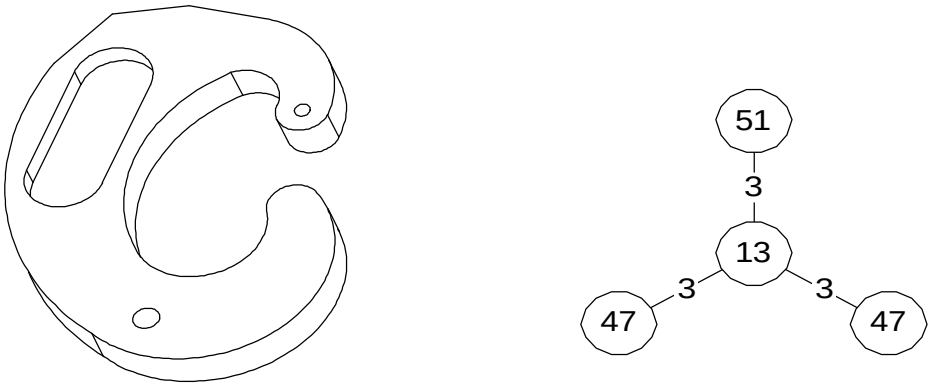


Figure 1.3: Graphe de géons. Le chiffre à l'intérieur du cercle identifie un géon particulier, alors que la valeur d'arc précise la relation spatiale existant entre deux géons.

Compte tenu des attributs symboliques associés aux géons, il est clair que la description par géons est foncièrement qualitative. Il s'agit d'une différence fondamentale avec les approches par modèles: la description par géons vise effectivement à désigner la famille à laquelle appartient l'objet, plutôt que l'individu précis. La description par géons est générique, dans la mesure où elle fait totalement abstraction des éléments qui particularisent les diverses instances d'une même famille.

Cet aspect générique des géons restreint d'ailleurs la portée de la théorie de *Reconnaissance par composantes* pour l'identification d'objets. Nous mentionnerons deux propriétés qui limitent le pouvoir discriminant des géons:

1. L'univers géométrique se projette dans le domaine des géons par une relation univoque plutôt que réciproque. Ainsi, à un cylindre généralisé donné correspond un et un seul géon, mais à un géon donné correspond une multitude d'instances de cylindres généralisés. La figure 1.4 illustre par exemple deux objets différents, mais pourtant décrits par le même géon.

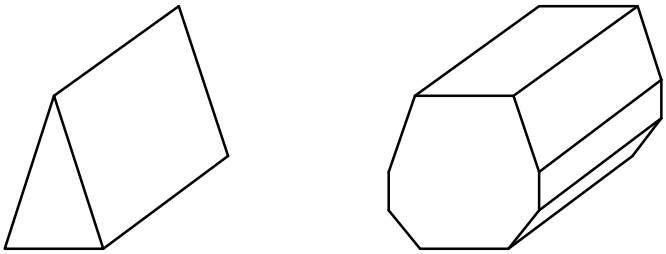


Figure 1.4: Géons semblables

2. La représentation par géons ne comporte aucune composante dimensionnelle. Ainsi, des objets morphologiquement similaires mais distincts par un seul facteur d'échelle auraient la même description géon, excluant toute discrimination précise.

Il faut noter finalement que si l'approche géon est inappropriée pour les applications de précision (métrologie, positionnement, etc.), elle pourrait par contre s'avérer particulièrement avenue comme clé de recherche dans une base de données: Biederman calcule en effet que plus de 154 millions d'objets qualitativement différents peuvent être différenciés à partir de toutes les combinaisons possibles de trois géons.

1.3 Attributs intrinsèques

1.3.1 Attributs des géons

Chaque géon sera spécifié par une séquence de 5 chiffres, présentés dans l'ordre suivant:

1. Matérialité du géon;
2. Courbure de l'axe;
3. Courbure des arêtes;
4. Symétrie du profil;
5. Fonction de balayage.

Les valeurs possibles de chaque attribut sont précisées dans le tableau 1.1. En plus des valeurs indiquées dans ce tableau, un attribut intrinsèque pourra aussi être représenté par une valeur nulle [Grégoire, 1998]. Ce dernier cas indique que la valeur d'attribut est inconnue.

Attribut	Nom	Valeurs	Significations
Type de géon	Matérialité (MT)	+1	Géon positif (solide)
		–1	Géon négatif (cavité)
Courbure de l'axe	Axe (AT)	0	Absence de connaissance
		1	Droit
		2	Incurvé
Arêtes du profil générateur	Arêtes (ET)	0	Absence de connaissance
		1	Droites
		2	Courbes
		3	Hybride: droites et courbes
Symétrie du profil générateur	Symétrie (S)	0	Absence de connaissance
		1	Asymétrie
		2	Symétrie réflective
		3	Symétrie réflective et rotationnelle
Fonction de balayage	Dimension (SD)	0	Absence de connaissance
		1	Constante
		2	Variable

Tableau 1.1: Valeurs d'attributs intrinsèques

1.3.2 Attribut de connectivité

L'attribut de connectivité pourra prendre cinq valeurs, mentionnées ci-dessous:

Valeur d'attribut	Position des axes	Type de jonction
1	Axes colinéaires	géons bout-à-bout
2	Axes à angle	jonction en T
3	Axes parallèles	côte-à-côte, inclusif, ...
4	Axes croisés	jonction en X
5	Axes confondus	géons concentriques

Tableau 1.2: Valeurs de l'attribut de connectivité

1.4 Répertoire de géons

Les cinq attributs intrinsèques engendrent une possibilité de 72 géons, listés ci-dessous. Les codes MT, AT, ET, S, SD abrègent le nom d'attribut indiqué au tableau 1.3 ci-dessous, alors que ID correspond à l'étiquette du géon.

ID	MT	AT	ET	S	SD
1	+1	1	1	1	1
2	+1	1	1	1	2
3	+1	1	1	2	1
4	+1	1	1	2	2
5	+1	1	1	3	1
6	+1	1	1	3	2
7	+1	1	2	1	1
8	+1	1	2	1	2
9	+1	1	2	2	1
10	+1	1	2	2	2
11	+1	1	2	3	1
12	+1	1	2	3	2
13	+1	1	3	1	1
14	+1	1	3	1	2
15	+1	1	3	2	1
16	+1	1	3	2	2
17	+1	1	3	3	1
18	+1	1	3	3	2
19	+1	2	1	1	1
20	+1	2	1	1	2
21	+1	2	1	2	1
22	+1	2	1	2	2
23	+1	2	1	3	1
24	+1	2	1	3	2
25	+1	2	2	1	1
26	+1	2	2	1	2
27	+1	2	2	2	1
28	+1	2	2	2	2
29	+1	2	2	3	1
30	+1	2	2	3	2
31	+1	2	3	1	1
32	+1	2	3	1	2
33	+1	2	3	2	1
34	+1	2	3	2	2
35	+1	2	3	3	1
36	+1	2	3	3	2

ID	MT	AT	ET	S	SD
37	1	1	1	1	1
38	1	1	1	1	2
39	1	1	1	2	1
40	1	1	1	2	2
41	1	1	1	3	1
42	1	1	1	3	2
43	1	1	2	1	1
44	1	1	2	1	2
45	1	1	2	2	1
46	1	1	2	2	2
47	1	1	2	3	1
48	1	1	2	3	2
49	1	1	3	1	1
50	1	1	3	1	2
51	1	1	3	2	1
52	1	1	3	2	2
53	1	1	3	3	1
54	1	1	3	3	2
55	1	2	1	1	1
56	1	2	1	1	2
57	1	2	1	2	1
58	1	2	1	2	2
59	1	2	1	3	1
60	1	2	1	3	2
61	1	2	2	1	1
62	1	2	2	1	2
63	1	2	2	2	1
64	1	2	2	2	2
65	1	2	2	3	1
66	1	2	2	3	2
67	1	2	3	1	1
68	1	2	3	1	2
69	1	2	3	2	1
70	1	2	3	2	2
71	1	2	3	3	1
72	1	2	3	3	2

1.5 Pièces de la base de données

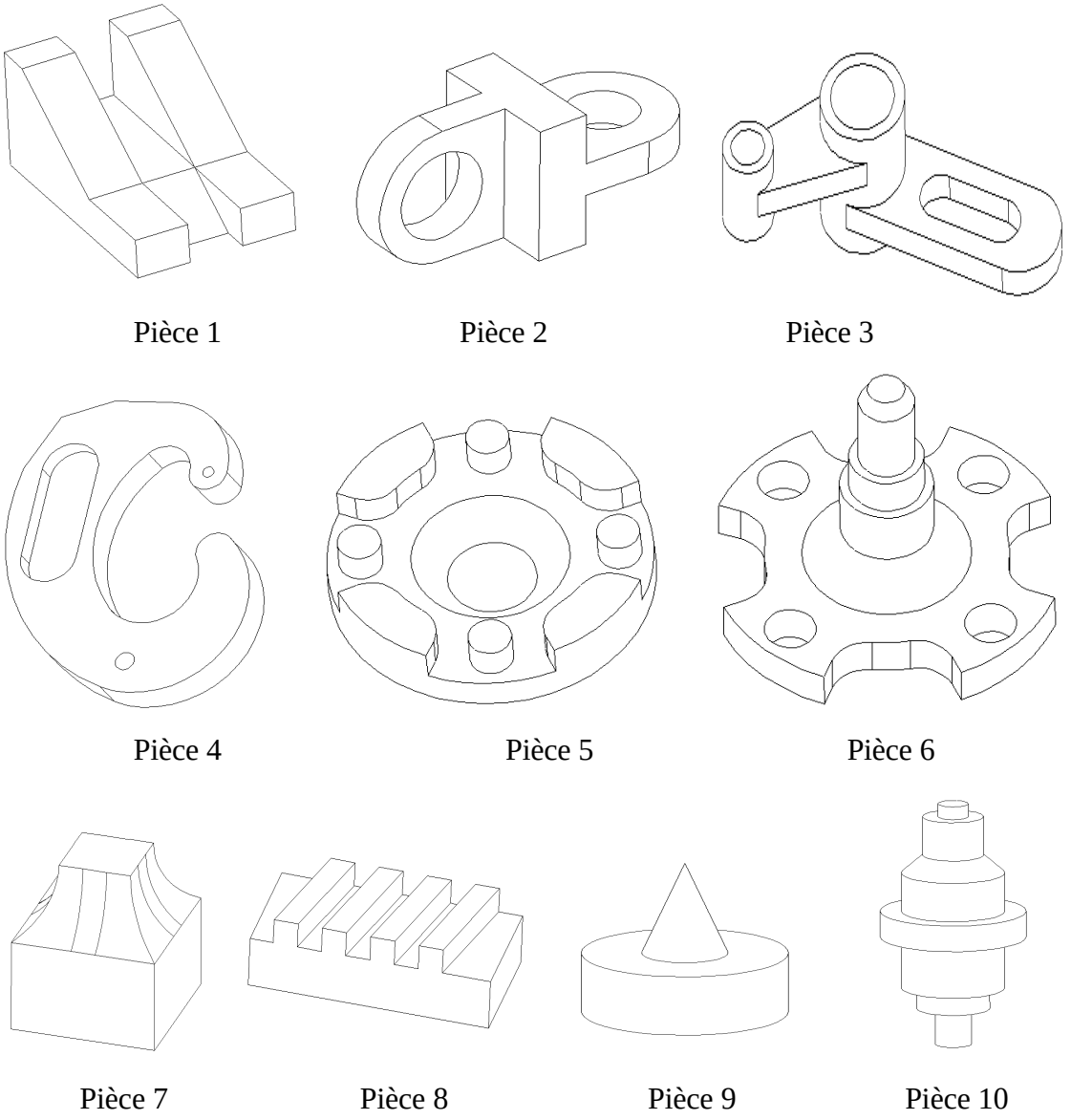


Figure 1.5: Répertoire de pièces manufacturées. La BD contient, pour chaque pièce, la représentation CAO (format neutre de type B-Rep produit par ProEngineer), leur image 3D mesurée au CNRC, leurs données 3D, les images 2D mesurées et leur représentation visuelle par géons.

1.6 Description géon des pièces

Les descriptions présentées ici constituent la référence du réseau neuronique et la clé d'indexation de la BD. Les géons sont indiqués par leur identificateur ID. L'ordre d'énumération des géons est arbitraire.

1.6.1 Géons constitutifs

pièce 1 1	1	1								
pièce 2 5	15	15	47	47						
pièce 3 11	11	15	15	47	47	51				
pièce 4 13	47	47	51							
pièce 5 11	11	11	11	11	15	15	15	15	48	
pièce 6 11	11	11	12	12	17	47	47	47	47	
pièce 7..... 5	6									
pièce 8 5	5	5	5	5						
pièce 9 11	12									
pièce 10 11	11	11	11	11	11	11	12			

1.6.2 Inventaire des géons utilisés

ID	Nature
1	forme positive asymétrique à arêtes droites
5	parallélépipède positif
6	forme positive à dimension variable
11	cylindre positif
12	cône positif
13	forme hybride positive asymétrique
15	forme hybride positive à symétrie réflective
17	forme hybride positive à symétrie totale
47	cylindre négatif
48	cône négatif
51	forme hybride négative à symétrie réflective

1.6.3 Description complète avec connectivité

Les pièces de la base de données sont décrites ici sous forme de graphes relationnels. Les nœuds de chaque graphe indiquent l'étiquette ID du géon, alors que les arcs expriment la valeur de l'attribut de connectivité.

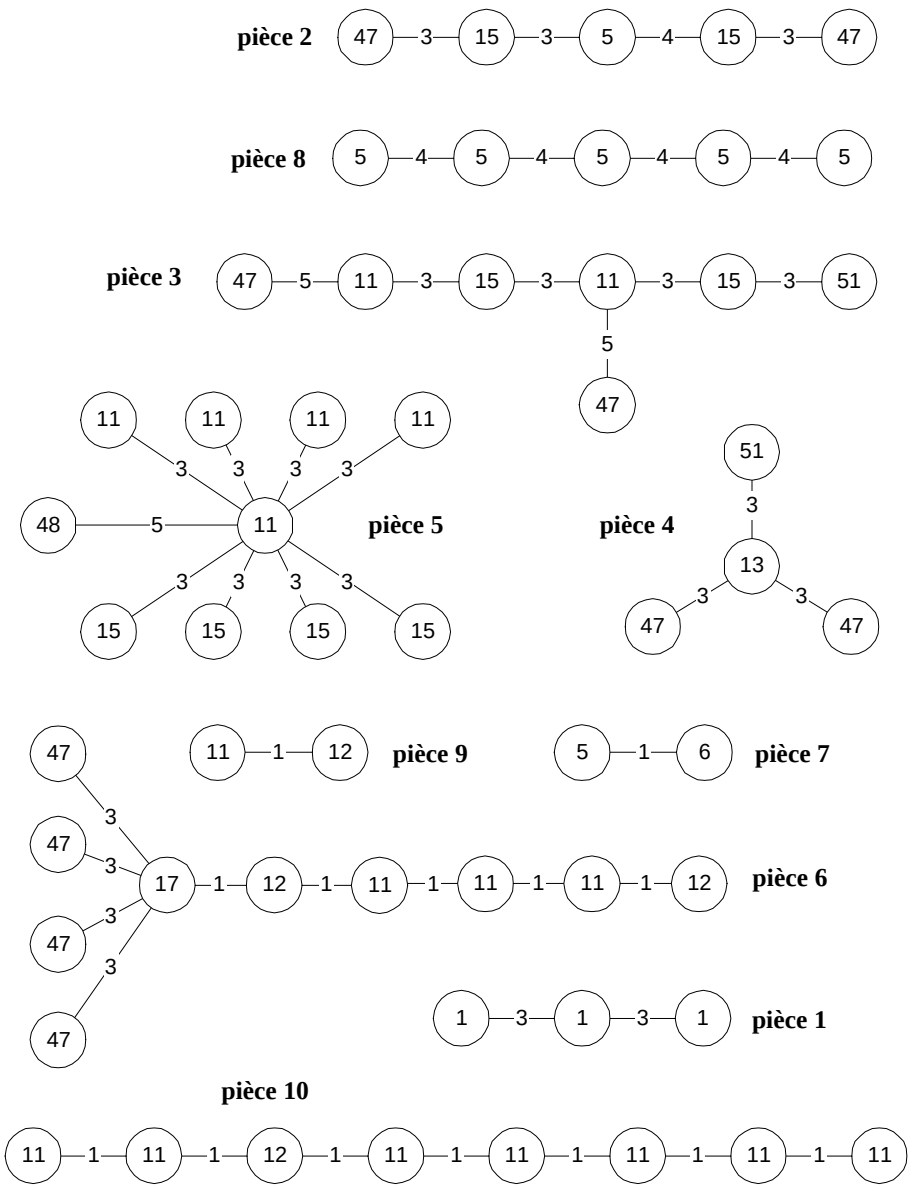


Figure 1.6: Description complète avec connectivité des pièces

1.7 Base de données

Notre base de données est centralisée. De ce point ses fichiers sont organisés en table d'adressage calculé (*Hashtable*) indexée dans le serveur *Hopfield* du LIVIA de l'ÉTS. Elle fait office d'un dépôt des données accessibles à tous les usagers autorisés. Ceux-ci peuvent accéder à des données de la base partagée par des opérations d'importation[3] ou y déposer des données par des opérations d'exportation [KIMWon et al., 1991]. Les usagers privilégiés peuvent maintenir sur leur station leurs propres bases de données privées avec les données qu'ils ont créées ou copiées de la base partagée.

1.7.1 Données associées à chaque pièce

À chaque pièce de la BD est associé:

- deux fichiers au minimum: description CAO de format (*neu* ou *iges*) et un icône visuel *.gif* utilisé pour visualiser une pièce de la base;
- une vue géon qui demande aucune ou plusieurs connexions entre un couple de 2 géons, comme elle peut avoir un ou plusieurs géons utilisés;
- Aucun ou plusieurs fichiers 2D (deux dimensions) de format *im*, *jpeg*, *tiff*, etc.;
- Aucun ou plusieurs fichiers 3D (trois dimensions) sous forme de nuages de points qui représentent l'image 3D de la pièce.

[3] Par exemple, en cas d'importation, un utilisateur autorisé peut accéder à notre base de données pour télécharger chez lui différentes types de fichiers concernant une pièce manufacturée. En cas d'exportation, un utilisateur autorisé peut accéder à notre base pour ajouter une nouvelle pièce manufacturée ou des fichiers supplémentaires associés à une pièce de la BD.

Le modèle CAO de format *iges* modélise la pièce en utilisant des surfaces planes. Il définit un modèle mathématique ou numérique (surfaces, dimensions) et représente le modèle idéal de la pièce. À partir de ce modèle, chaque pièce est inspectée.

Le modèle CAO en format *neu* décrit la pièce en utilisant un langage propre au modélisateur CAO. Il est utilisé pour la description des géons.

Une vue 2D représente l'image de la pièce à 2 dimensions (X,Y). Notons qu'on peut voir la même pièce sous un autre point de vue et on aura deux images 2D différentes illustrées dans la figure 1.7 suivante.

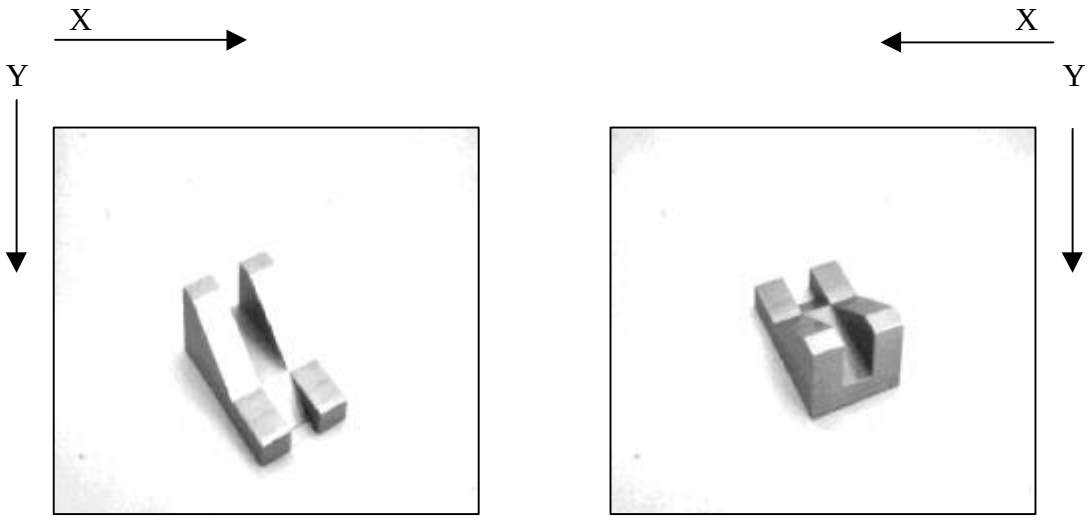


Figure 1.7: Vue à deux dimensions d'une pièce de la base

Le prochain chapitre est consacré à la modélisation objet proposée pour la base de données: il introduit la base de données, précise ses besoins et se termine avec le modèle objet qui sera utilisé dans la réalisation du projet d'inspection.

CHAPITRE 2

MODÉLISATION ORIENTÉE-OBJET DE LA BASE DE DONNÉES

Ce chapitre illustre la méthode d'analyse proposée pour le projet, il clarifie à partir des définitions les principaux concepts de l'analyse orientée-objet (AOO). Aussi, il précise le langage de programmation et son environnement. Les besoins et les objectifs qui sont associés à la base de données d'inspection sont introduits au moyen des concepts de base pour les systèmes de gestion de base de données (SGBD). Les fichiers qui composent la base de données sont décrits dans ce chapitre. Ce chapitre se termine avec la modélisation orientée-objet, qui sera utilisée pour la réalisation de la base de données.

2.1 Pourquoi avons-nous besoin de l'analyse orientée-objet?

Les raisons clés et les avantages de l'AOO sont présentés ici pour guider deux catégories de lecteurs. Pour le responsable de projet, cette section explique pourquoi il devrait encourager les membres de son équipe à utiliser l'AOO. Pour un membre d'une équipe de projet, cette section présente un argumentaire qu'il pourra utiliser pour convaincre son responsable de la pertinence de l'AOO.

Coad et Yourdon (1992) ont développé une AOO dont les concepts ont été puisés des modélisations sémantiques de données (*information modeling*), des langages orientés-objet et des bases de connaissances.

Les raisons et les avantages de l'AOO sont les suivants:

1. Aborde mieux les questions propres aux domaines du problème. L'AOO met l'accent sur la compréhension des divers domaines du problème.

2. Améliore l'interaction entre l'expert du domaine du problème et l'analyste. L'AOO organise l'analyse et la spécification en utilisant des méthodes d'organisation plus proches de la pensée des individus.

3. Augmente la cohérence interne des résultats de l'analyse. L'AOO réduit la séparation entre les différentes activités d'analyse, en traitant les attributs et les méthodes comme un tout intrinsèque.

4. Représente explicitement les éléments communs. L'AOO utilise l'héritage pour identifier et capitaliser sur les éléments communs des attributs et des méthodes.

5. Assure une stabilité lors des changements des besoins et lors de la construction de systèmes similaires.

6. Réutilise des classes au niveau du code, du design et de l'analyse, ce qui représente le bénéfice principal de l'orientation objet. Elle se révèle comme la meilleure stratégie pour:
   - maîtriser rapidement les difficultés de développer des systèmes logiciels complexes,
   - bâtir des systèmes de meilleure qualité,
   - améliorer la productivité des concepteurs et des développeurs de logiciels d'application.

7. Fournit une représentation de base cohérente pour l'analyse (ce qui doit être construit) et la conception (comment cela doit être construit à un moment donné). L'AOO définit un continuum de représentation pour transformer systématiquement les résultats de l'analyse en une conception spécifique.

La représentation graphique du modèle orientée-objet a été réalisée pour ce travail avec l'outil: *Playground.exe*, version 2.0.1. Cette représentation inclut les quatre couches du modèle de l'AOO et leurs symbolismes (cf. figure 2.1 Représentations graphiques de l'AOO).

• Couche classe&objets
  Classes&objets
  Classes

• Couche structure
  Structure de Généralisation-spécialisation (Héritage)
  Structure de Composé-composant (Fait partie)

• Couche attribut
  Attributs
  Connexions d'instances (Relation)

• Couche Méthode
  Méthodes
  Connexion de messages[1]

[1] Une connexion de messages modélise la dépendance des traitements d'un objet indiquant un besoin en service lui permettant de remplir ses obligations.

Une méthodologie comprenant ces étapes d'activités inter-reliées est fournie pour couvrir l'étude du domaine du problème selon les couches ci-dessous:

1- Trouver les objets et les classes.
2- Identifier les structures.
3- Définir les attributs.
4- Définir les services.

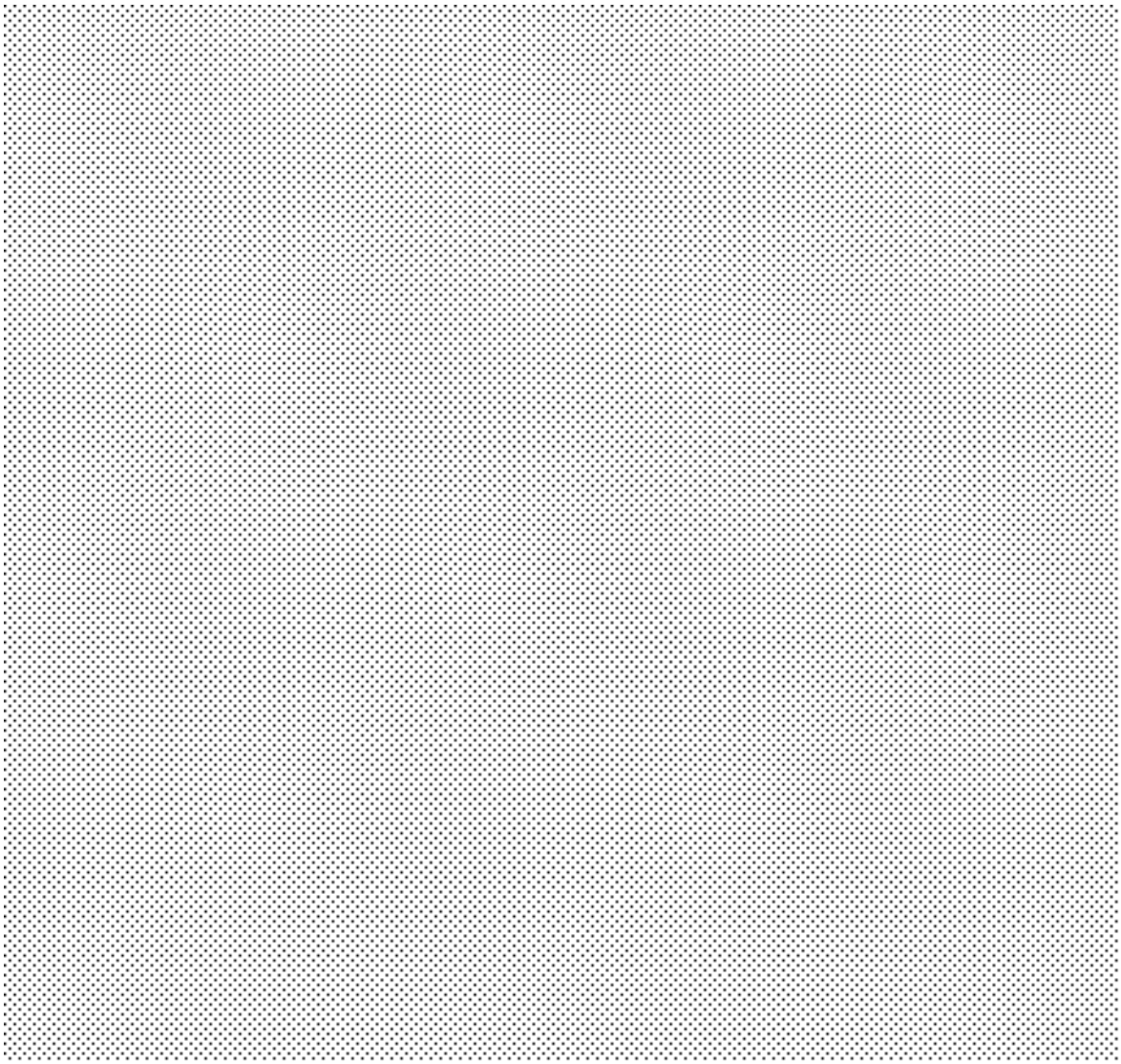


Figure 2.1: Représentations graphiques de l'AOO

2.2 Définitions

2.2.1 Objet

Un objet est un concept, une abstraction ou une chose ayant des limites très claires et un sens précis dans le contexte du problème étudié. Chaque objet a une identité et peut être distingué des autres.

2.2.2 Classe et instance

En Orientée-Objet, une classe décrit un ensemble d'instances d'objets d'une forme donnée. L'instantiation relie une classe à ses instances. La figure 2.2 montre un exemple d'instantiation. *Pièce #1* et *Pièce #2* sont des instances de la classe *Pièce*. Les flèches en pointillés relient les instances à la classe [Rumbaugh et al., 1996].

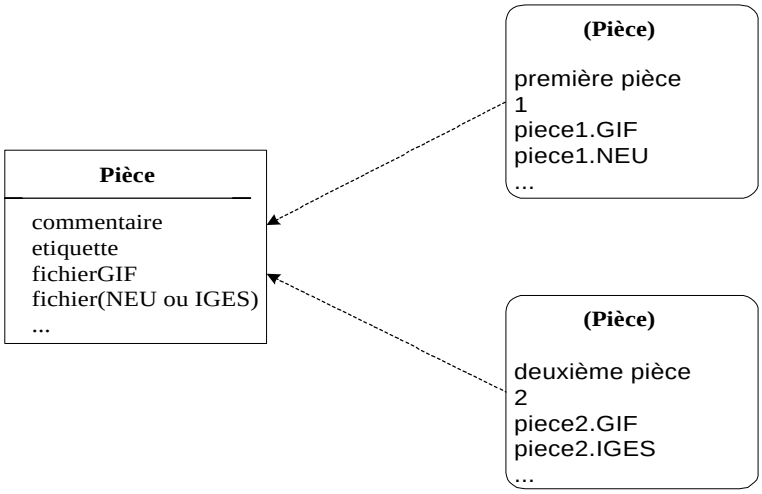


Figure 2.2: Notation pour L'instantiation

Ainsi, une classe est un modèle définissant les méthodes et les variables pour un type particulier d'objet ayant des caractéristiques similaires [Taylor, 1993]. Dans l'exemple présenté à la figure 2.2, *Pièce #1* et *Pièce #2* sont des objets distincts mais ayant des propriétés communes comme pièces de la base de données d'inspection.

Chaque objet appartient à une classe. Un objet appartenant à une classe donnée est appelé instance de cette classe. Pendant que la classe décrit la structure de l'instance, l'instance d'une classe ne contient que les valeurs propres des attributs de l'objet. Il est bon de noter que dans le monde de l'orienté objet, les termes instance et objet sont souvent utilisés comme s'ils étaient synonymes [Hess, 1993; Jacobson et al., 1993].

Les classes permettent de définir des objets de manière efficace. Les méthodes et les variables d'une classe ne sont définies qu'une seule fois, lors de la définition de la classe, sans que l'on ait à les répéter pour chaque instance de la classe.

2.2.3 Attribut

En Analyse Orientée-Objet (AOO), le terme *attribut* est défini pour refléter le domaine du problème et les responsabilités du système. Un attribut est une donnée (information d'état) pour laquelle chaque objet dans une classe a une valeur propre.

En Java, les attributs sont définis dans les classes par des variables. Les types et les noms de ces variables sont définis dans la classe. Chaque objet d'une classe pouvant avoir des valeurs différentes pour ses variables, on utilise le terme *variable d'instance* pour désigner les variables d'un objet. Une variable d'instance définit un attribut d'un objet. Les types et les noms des variables sont définis dans la classe, mais leurs valeurs sont fixées et modifiées dans l'objet.

2.2.4 Méthode

En AOO, le terme *méthode* est défini pour refléter le domaine et les responsabilités du système. Une méthode est un comportement spécifique qu'un objet est en charge de fournir. Elle est équivalent à une fonction en programmation procédurale.

2.2.5 Constructeur

En programmation Orientée-Objet (POO), chaque classe contient au moins une méthode appelée constructeur. Cette méthode porte le même nom que la classe qui la contient et ne doit pas indiquer de valeur de retour; l'objet retourné est implicitement une instance de la classe.

En java, avec le *new nom de la classe( )*, le noyau crée physiquement l'objet correspondant (il réserve de la place en mémoire). Pour chaque classe, une ou plusieurs méthodes particulières peuvent être définies: ce sont les constructeurs. Le constructeur est appelé au moment de la création de l'objet. L'utilisation de *new( )* entraîne en effet la création physique de l'objet et l'appel d'un des constructeurs.

2.2.6 Connexion d'instance

Une connexion d'instances est un modèle des liaisons du domaine du problème qu'un objet doit avoir avec d'autres objets dans le but de remplir ses obligations.
(cf. figure 2.7: la connexion d'instance *Peut avoir* entre les objets: Pièce et Fichier).

2.2.7 Message et polymorphisme

Un message est une requête qui demande à l'objet récepteur d'exécuter la méthode indiquée et de renvoyer le résultat de cette exécution à l'objet requérant. La figure 2.3 illustre ce processus.

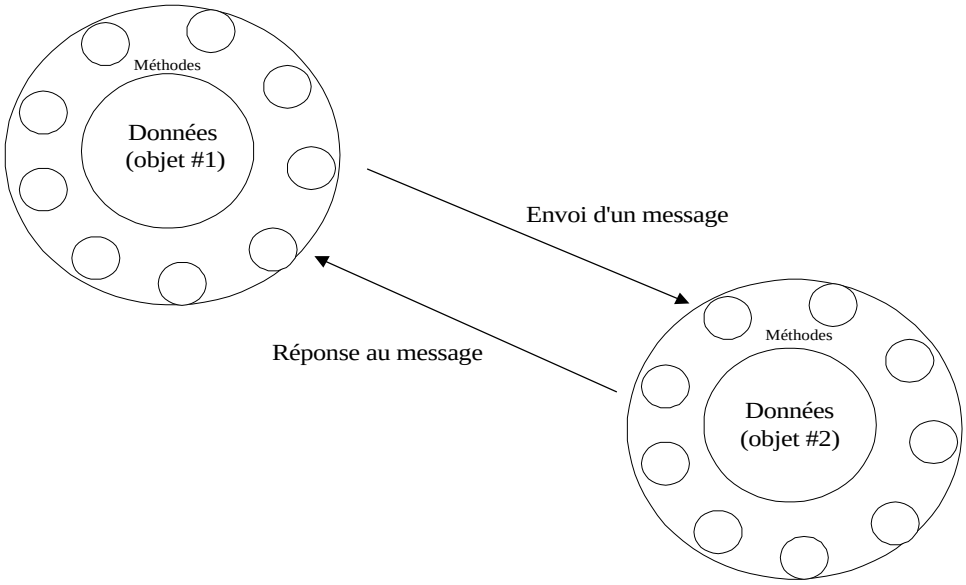


Figure 2.3: Les messages

Une même méthode peut s'appliquer à un nombre quelconque d'objets, et chacun d'entre eux appliquera cette méthode selon des besoins qui lui sont propres. Cela permet d'envoyer un message à plusieurs objets différents sans avoir à s'inquiéter de la manière dont le message sera traité, et ce sans même connaître le type d'objet qui le recevra. Cette faculté de dissimuler les détails d'implémentation derrière une interface commune de message s'appelle polymorphisme [Taylor, 1993].

2.2.8 Structures Composé-Composants

D'après Coad et Yourdon (1992), Il faut considérer chaque objet comme un composé. Pour ses composants potentiels, il faut se poser les questions suivantes:

• Sont-ils dans le domaine du problème?
• Sont-ils dans les limites des responsabilités du système?
• Est-ce qu'ils prennent en compte autre chose que la valeur d'un état?
• Est-ce qu'ils fournissent une abstraction utile qui traite bien du domaine du problème?

Il faut considérer, de la même manière, chaque objet comme un composant.
Pour chaque composé potentiel, il faut se poser les mêmes questions. (cf. figure 2.7: la relation *Fait partie* entre les objets: Connexion et Attribut).
Le composé représente l'objet «Connexion» et le composant représente l'objet « Attribut ».

2.2.9 Structures Généralisation-Spécifications

Il faut considérer chaque classe comme une généralisation. Pour ses spécialisations potentielles, il faut se poser les questions suivantes [Coad et Yourdon, 1992]:

• Sont-elles dans le domaine du problème?
• Sont-ils dans les limites des responsabilités du système?
• Y a-t-il héritage?

2.2.9.1 Notion d'héritage

L'avantage de l'héritage dans un langage de programmation par objets est que le concepteur/réalisateur peut dire Oui, je voudrais réutiliser ce que vous avez là, sauf que j'aimerais ajouter ces variables et ajouter ou élargir les traitements de cette manière . Ce comportement est tout à fait différent de celui qui consisterait à dire: Ok. Ceci est ce que vous avez développé, et je dois l'utiliser tel quel ou sinon je vais devoir l'écrire moi-même , (typique en programmation procédurale).

L'héritage est un mécanisme important pour la réutilisation et pour saisir explicitement les éléments communs [Coad et Yourdon, 1992]. Il inclut quatre aspects principaux:

1. Chaque classe peut hériter des variables définies dans sa(ses) superclasse(s).
2. Chaque classe peut hériter des méthodes définies dans sa(ses) superclasse(s).
3. Chaque classe peut ajouter des variables.
4. Chaque classe peut ajouter ou élargir les méthodes héritées.

(cf. figure 2.7: la relation *Est héritée* entre les objets: Service et Fil (*Thread*)).

De plus, l'héritage renforce l'efficacité du mécanisme de classe en permettant de ne définir le comportement similaire d'un grand nombre d'objet qu'une seule fois à l'intérieur de la classe de plus haut niveau [Taylor, 1993]. Les sous-classes ne feront que modifier le comportement pour décrire un cas particulier.

2.2.10 Modélisation

La modélisation est l'activité qui consiste à bâtir des modèles, soit pour décrire un système d'information existant (analyse), soit pour élaborer un nouveau système d'information (conception).

2.3 Langages de programmation et environnement de développement

Pour réaliser le développement des deux logiciels (un pour la partie client et un pour la partie serveur), ainsi que pour interroger la base de données à distance, on a utilisé le *JAVA* comme langage de programmation avec l'environnement de développement *JDK* (*Java Developer Kit)*.

2.3.1 Pourquoi JAVA ?

- **Java ne dépend d'aucune plate-forme**

Son indépendance vis-à-vis des plates-formes matérielles est son principal avantage. Il est donc tout indiqué pour les applications qui doivent tourner sur différentes plates-formes. Cette caractéristique est vitale pour tout logiciel destiné au Word Wide Web.

L'indépendance, dans le cas de Java, se situe à deux niveaux: source et binaire. Au niveau source, les types de données primaires de Java ont la même taille, quelle que soit la plate-forme de développement. Ainsi, avec le compilateur, le programme Java peut être traduit en langage intermédiaire binaire appelé *bytecode.* Les bibliothèques de classes standard de Java facilitent l'écriture du code, qui est ensuite porté de plate-forme en plate-forme, sans adaptation [Macary et al., 1996].

- **Java est Orienté-Objet**

Pour certains, la programmation Orientée-Objet (POO) s'assimile à l'organisation des programmes, ce qui est réalisable avec tous les langages. Néanmoins, travailler avec un langage et un environnement de programmation Orienté-Objet permet de tirer le meilleur parti de cette méthodologie, qui permet de créer des programmes souples et modulaires et d'en réutiliser le code.

Comme la plupart de ces langages, java possède également des bibliothèques de classes, qui fournissent les types de données de base, les possibilités d'entrées/sorties du système et d'autres fonctions utilitaires. Ces bibliothèques de base font partie de JDK, qui comprend également des classes pour la gestion de réseau, des protocoles Internet communs et des outils d'interface utilisateur. Ces bibliothèques sont écrites en langage java, et sont donc portables sur toutes les plates-formes.

2.3.2 Environnement de développement adapté à notre plate-forme

Pour programmer en java, on a besoin d'un environnement de développement java adapté à notre plate-forme. Le JDK de Sun répond à cette exigence. Il comporte des outils de compilation et de test des applets et des application java [Desfray, 1997].

D'une manière générale, un environnement de développement orienté objet fournit des classes génériques (*class librairies*) qui contiennent des connaissances dont la qualité a été testée, et qui sont prêtes à être héritées et réutilisées. Les classes définies par les utilisateurs viennent aussi enrichir progressivement cet environnement de développement.

Le JDK de Sun javasoft

Le java Developer Kit est l'environnement de base minimum pour effectuer des développements java. Il n'est disponible aujourd'hui que pour les plates-formes Sparc, Solaris, Windows95 et Windows NT.

Il contient:

- Les bibliothèques de classes standards (compilées et sources) suivantes: java.util, java.io, java.net, java.awt, etc.;
- Le compilateur (javac), produisant le p-code;
- Un générateur de documentation (javadoc), etc.

- javac

javac compile le source java. Les sources java doivent être enregistrés dans des fichiers de suffixe .java; javac analyse ce fichier .java et produit autant de fichiers .class qu'il y a de classes dans le fichier .java . Il n'y a donc pas forcément le même nombre de fichiers .class que de fichiers .java.

Parmi les classes présentés dans le fichier .java, il ne peut y avoir qu'une seule classe dite publique. Cette classe doit porter le même nom que le fichier.

- java

L'outil java prend en paramètre le nom d'une classe. Il va donc chercher un fichier .class correspondant à ce nom, charger la classe correspondante et appeler la méthode *main* de cette classe. S'il ne la trouve pas, il signale l'erreur et abandonne.

- javadoc

javadoc sert à générer automatiquement la documentation html des classes que vous développez. C'est avec cet outil que la documentation sur les *API*[2] des bibliothèques fournies avec java a été générée.

- package

Les *packages* regroupent des classes et des interfaces ayant un rapport entre elles, dans une seule collection. Grâce à eux, les groupes de classes sont disponibles uniquement s'ils sont nécessaires, ce qui permet notamment d'éviter des conflits entre noms de classe de différents groupes. Par convention, les noms du package commencent plutôt par une lettre minuscule pour les distinguer des noms de classe.

[2] *API* (*Application Program Interface*), décrit l'ensemble des fonctions et des procédures disponibles pour le langage JAVA.

2.4 Introduction à la base de données

Notre base de données est conçue pour traiter de gros volumes d'informations concernant les pièces manufacturées. Ainsi elle est constituée d'un ensemble de fichiers contenant chacun un type particulier de données:
fichier *ListePiece,* fichier *ListeAttributGeon,…*

Une base de données désigne une collection d'articles comportant deux informations essentielles: les données elles-mêmes (détails des pièces, géons, …) et les relations entre les articles contenus [Gunton, 1993].

La gestion de données impose une structuration de ces données ainsi qu'une rationalisation des procédures d'accès et de traitement qui constituent le logiciel d'exploitation. On doit aussi assurer la sécurité des informations stockées dans la base. En cas d'exploitation partagée entre plusieurs utilisateurs, il est nécessaire d'éviter d'éventuelles situations conflictuelles [Elmasri, 1989].

2.4.1 Avantages et inconvénients

L'un des avantages clés de la gestion de base de données est qu'il permet à différents laboratoires d'inspection d'utiliser les mêmes données concernant des pièces telle qu'illustré dans la figure 2.4. Malheureusement, cette disponibilité présente aussi des inconvénients au niveau de la sécurité. Pour éviter les problèmes sécuritaires, le gestionnaire de projet d'inspection automatisée des pièces manufacturées s'appuie sur des mécanismes de sécurité qui font partie intégrante du SGBD et limitent l'accès à certains enregistrements à ceux qui en ont l'autorisation. Chaque utilisateur (local ou à distance) doit disposer d'un code d'identification et d'un mot de passe qui définit un niveau d'accès particulier à la base de données d'inspection.

2.4.2 Vue globale du système d'inspection automatisée

Notre système est composé d'un ensemble d'utilisateurs répartis géographiquement qui accèdent à une base centralisée telle qu'illustré ci-dessous.

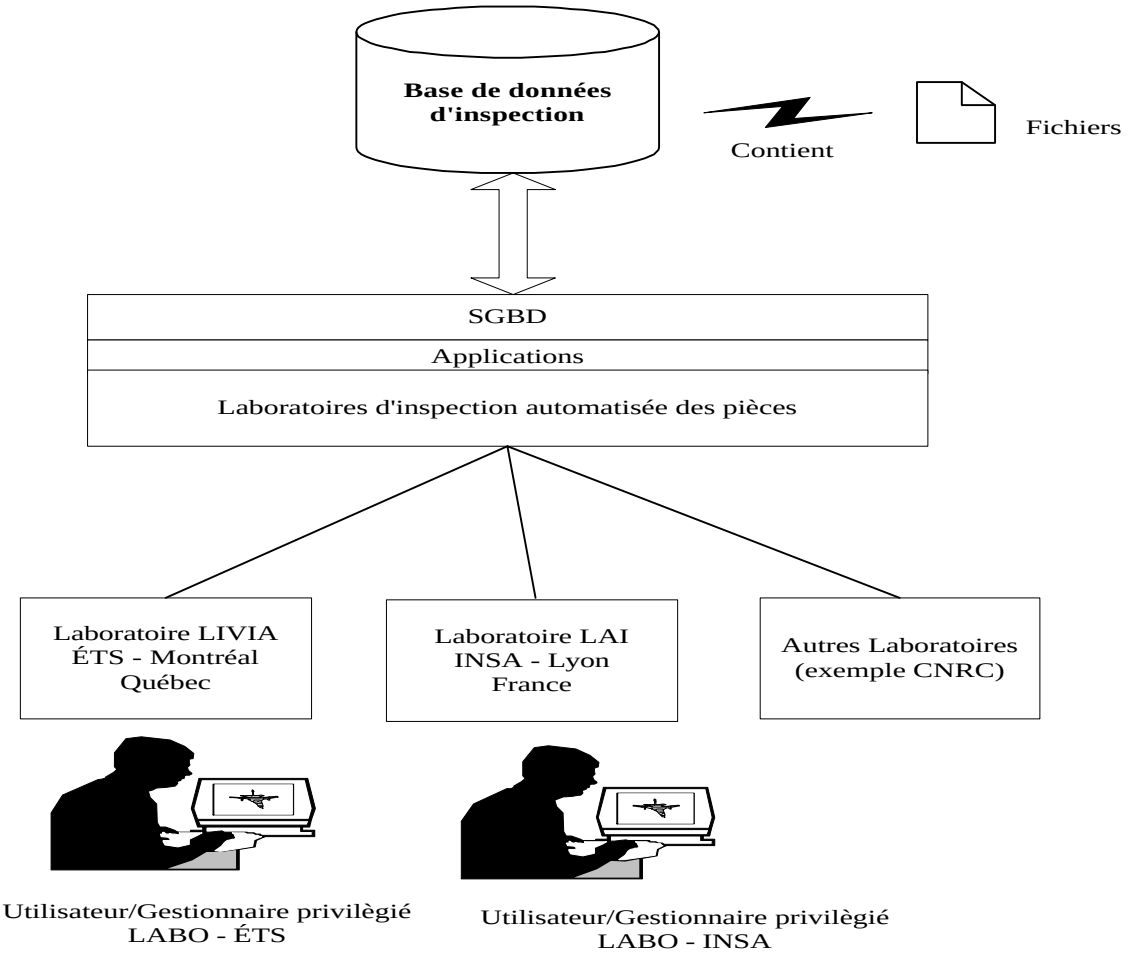


Figure 2.4: Application partageant la base d'inspection

Les données de la base d'inspection sont stockées sur le serveur *hopfield* de l'ÉTS sous forme de fichiers binaires et elles sont partagées entre plusieurs utilisateurs français et canadiens. Le principal rôle du SGBD consiste à gérer les différents accès à la base de données.

2.4.3 Besoins pour la base de données d'inspection

La réalisation du projet "*Inspection automatisée de pièces manufacturées*" nécessite une structuration d'une base de données centralisée[3] pouvant contenir toutes les informations à propos des pièces et le développement des logiciels permettant d'interroger la base. Le développement de deux outils (un pour la partie serveur et un pour la partie client) était nécessaire afin de pouvoir interroger à distance la base de données.

La partie serveur contient la base de données proprement dite avec toutes les informations stockées sous forme de fichiers binaires. Cette partie rassemble aussi tous les outils permettant la mise à jour de la base.

La partie client est constituée d'un programme permettant de questionner le serveur à distance dans le but de visualiser ou de compléter la base de données et ceci grâce à une interface[4] graphique simple à utiliser.

2.4.4 Buts principaux associés à la base de données

Les principaux objectifs associés à la base sont les suivants:

1) Stocker/Mettre à jour les informations sur des pièces mécaniques.
2) Consulter la base de données.
3) Modifier la base de données.
4) Recevoir les requêtes envoyées des clients à partir de la douille de communication.

---

[3] L'architecture centralisée consiste en un groupe central fort autour duquel tous les périphériques sont regroupés (ou centralisées). Elle prône un noyau central alors que l'architecture distribuée en prône plusieurs.

[4] Nous référons le lecteur à l'annexe A.

5) Construire une base de données d'inspection qui est une collection d'éléments de la base de données, c'est-à-dire échantillonner quelques pièces pour un projet d'inspection spécifique (cf. Annexe F: fenêtre ″Ajouter une base réduite″, page 129).

À bien remarquer que la mission principale de la base de données à l'ÉTS (LIVIA) ou dans un centre de recherche, consiste à pouvoir extraire une base de données restreinte[5] qui sert à manipuler les pièces manufacturées.

2.5 Système de gestion de base de données (SGBD)

2.5.1 Concepts généraux des SGBD

Un système de gestion de base de données est un programme informatique permettant de gérer un entrepôt de données. [Rumbaugh et al., 1996]. Cet entrepôt de données est appelé base de données et est stocké dans un ou plusieurs fichiers (cf. figure 2.4 précédente). De nombreuses raisons poussent les développeurs à employer des SGBD:

- *Reprise sur panne*. La base de données est protégée contre les fautes matérielles, les pannes de disques et certaines erreurs de l'utilisateur.
- *Partage entre utilisateurs*. Comme la base de données d'inspection est partagée entre plusieurs laboratoires français et canadiens, plusieurs utilisateurs peuvent accéder à cette base en même temps.
- *Partage entre application*. De multiples programmes d'applications peuvent lire et écrire les données dans la même base.
- *Sécurité*. Les données peuvent être protégées contre des accès non autorisés en lecture/écriture.

[5] Base de données, fonction des besoins de chaque client. Exemple: lors du contrôle dimensionnel d'une pièce, un client n'a besoin que du modèle CAO (IGES) de la pièce.

- *Intégrité*. Vous pouvez spécifier des règles à respecter par les données. Un SGBD peut contrôler la qualité de ses données plus avant et au-delà des moyens offerts par les programmes d'application.
- *Extensibilité*. Des données peuvent être ajoutées à la base sans déranger les programmes existants. Les données peuvent être réorganisées pour augmenter la rapidité.
- *Distribution des données*. La base de données peut être partitionnée sur des sites variés, sur diverses entreprises et diverses plates-formes matérielles. (C'est le cas d'une base de données distribuée, selon laquelle les données sont stockées sur plus d'un site).

2.5.2 Principaux modèles sur lesquels s'appuient les SGBD

Il existe 4 modèles principaux sur lesquels s'appuient les systèmes de gestion de base de données (SGBD):

- Le modèle hiérarchique.
- Le modèle réseau.
- Le modèle relationnel.
- Le modèle objet reposant sur les concepts des langages de programmation à objets (Java, C++, …). Ce dernier est utilisé dans notre projet.

Dans les langages associés aux SGBD on trouve:

- Un langage de définition de données (LDD), permet de définir dans le SGBD les objets représentés à l'aide du modèle de données: définition des objets, définition des relations, etc.

- Un langage de manipulation de données (LMD), permet de manipuler les données définies et chargées dans la base: sélectionner, insérer, supprimer, modifier, etc.

2.6 Les systèmes de gestion de fichiers (SGF)

2.6.1 Définition d'un fichier

Folk et Zoellick (1992) définissent un fichier comme étant un ensemble d'information stockées en mémoire secondaire. Il est utilisé pour: stocker, rechercher et transférer des données. Chacune de ces opérations nécessite un temps de traitement et un volume de support de stockage.

2.6.2 Les fichiers de la base de données

Notre base d'inspection contient les fichiers suivants:

ListePiece: Ce fichier contient toutes les pièces stockées dans la base de données.

ListeAttributConnexion: Ce fichier contient toutes les valeurs des attributs existantes pour les connexions.

ListeAttributGeon: Ce fichier contient toutes les valeurs des attributs existantes pour les géons.

GeonExistant: Ce fichier contient tous les géons qu'il est possible d'utiliser.

TypeFichier: Ce fichier contient tous les types existants de fichiers (gif, iges, neu, im, jpeg, prt, stl, tiff, ras, ...).

BaseReduite: Ce fichier contient la liste des bases d'inspection déjà créées.

2.6.3 Objectifs des SGF

1) Stocker efficacement les données. En effet, mieux on range l'information plus vite on la retrouve. On va donc trier l'information, l'étiqueter, définir des structures de données pour la localiser: les index.

2) Gérer efficacement la mémoire secondaire et permettre de:

Créer un fichier.

Ouvrir un fichier.

Lire un fichier.

Écrire dans un fichier.

Détruire un fichier.

2.6.4 Organisation de fichiers

Dans notre projet, la stratégie de rangement des enregistrements dans un fichier est sous forme de table d'adressage calculé (*Hashtable*). Cette dernière est indexée. La table d'adressage calculé est composé d'éléments, chaque élément contient une clé (index) et un contenu. On peut illustrer la table d'adressage calculé contenant les fichiers d'une pièce comme ci-dessous:

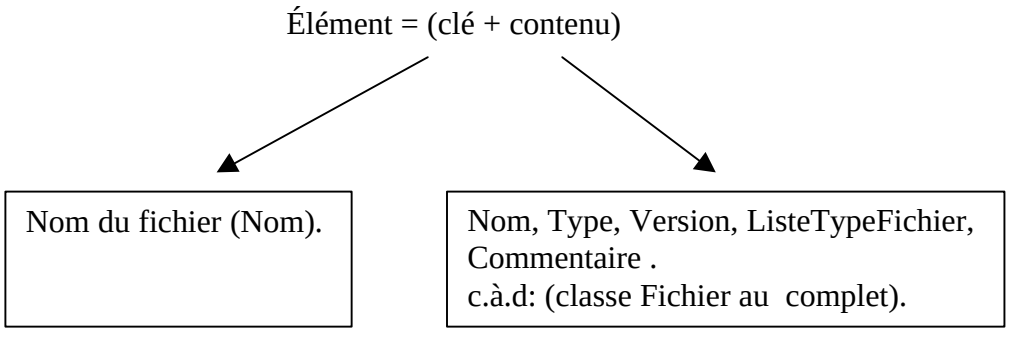


Figure 2.5: Organisation de fichiers de la base de données

À bien noter que, pour accéder à un élément, il faut connaître sa clé.

2.7 Modèle conceptuel de données (MCD)

Dans ce modèle, les données de la base d'inspection se présentent à l'aide d'un diagramme entité-relation dont les principaux composants sont les entités, les relations et les propriétés [Gabay, 1993].

Les entités, dessinées sous forme de rectangles, représentent différents ensembles d'individus, de choses ou d'événements qui font partie du domaine.

Les relations dessinées sous forme de figures arrondies, représentent différents ensembles d'interaction qui existent entre des entités et qui font partie du domaine. Une relation relie un certain nombre d'entités (au moins deux) qui sont ses entités participantes. Sa signification s'obtient en formant une phrase avec le verbe qui la désigne et les noms des entités qui lui sont reliées: la flèche placée près de la relation indique le sens de la lecture entre deux entités.

Une relation entre deux entités est caractérisée par:

- cardinalité minimale (égale à 0 ou 1): représente le nombre de fois minimum qu'une occurrence d'un objet participe aux occurrences de la relation.
- cardinalité maximale (égale à 1 ou N): indique le nombre des fois maximum qu'une occurrence de l'objet participe aux occurrences de la relation.

Le modèle conceptuel de données est illustré dans la page suivante. À partir de ce modèle, la modélisation orientée-objet a été bâtie.

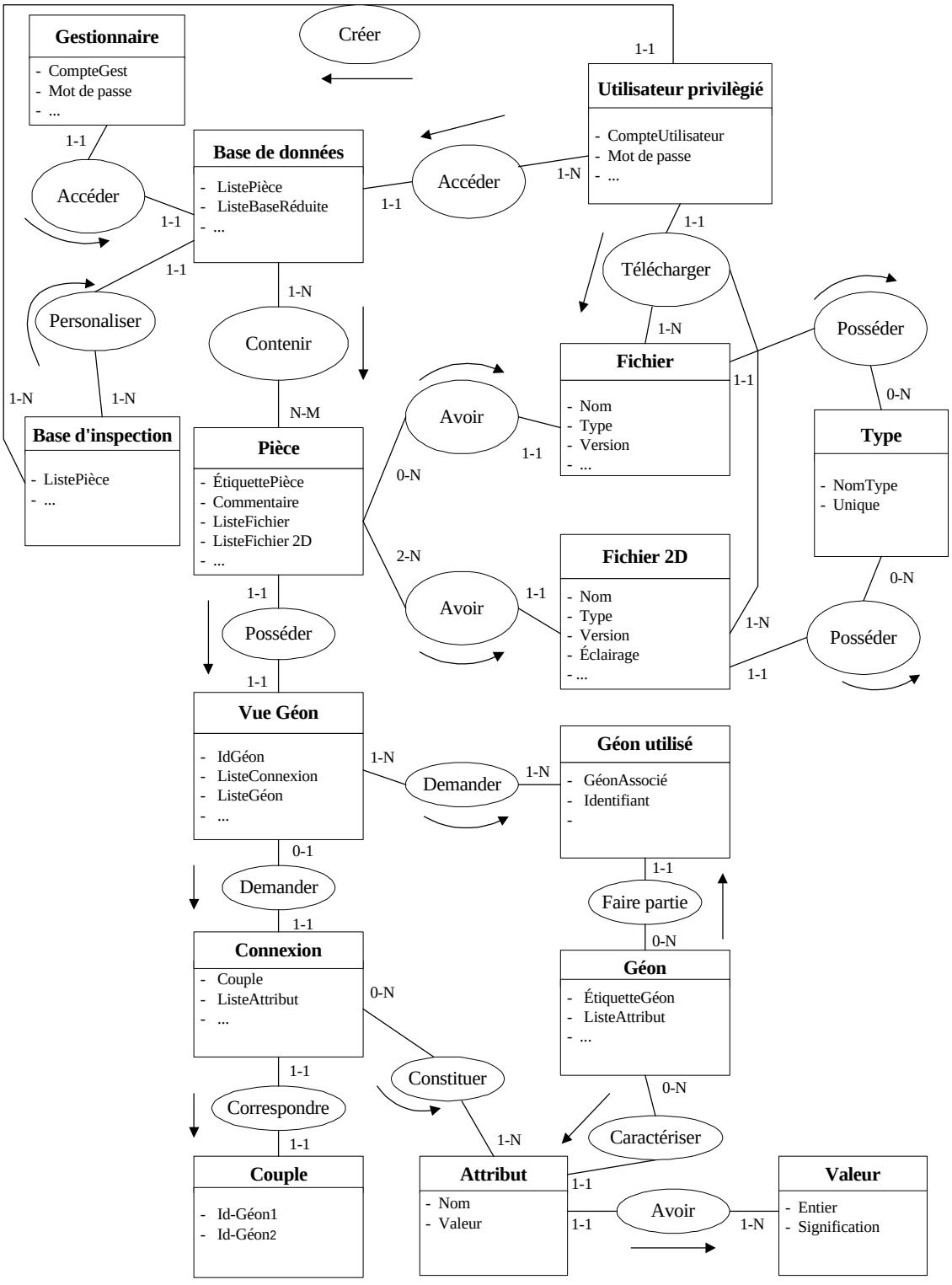


Figure 2.6: Le modèle conceptuelle de données (MCD)

2.8 Modélisation Orientée-Objet

Le but de l'analyse est de développer un modèle de ce que le système doit faire. Ce modèle est exprimé en termes de classes d'objets et de leurs relations [Davis, 1993]. Les données de la base sont arrangées de la manière suivante:

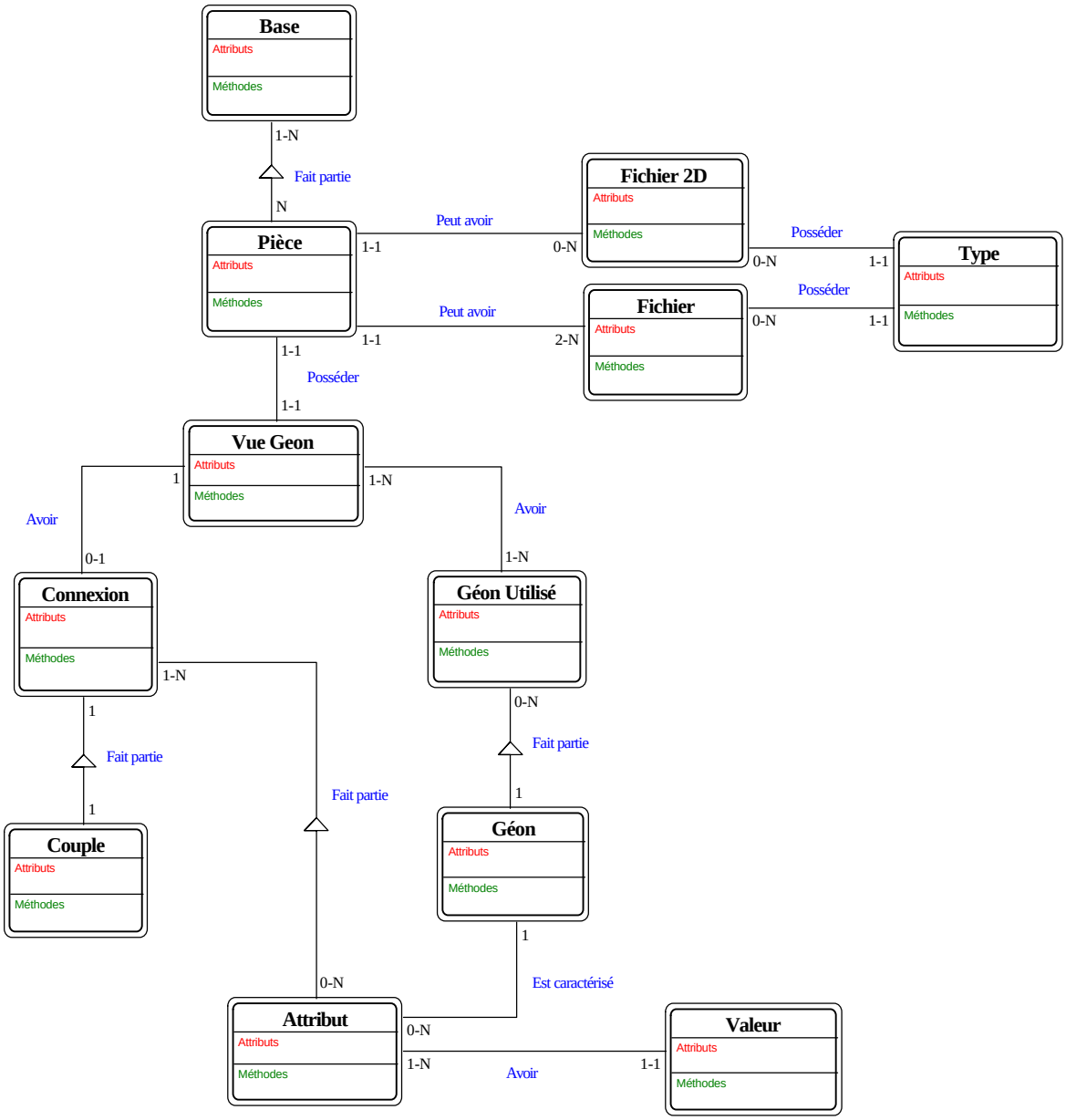


Figure 2.7: Le modèle Orienté-Objet

(Suite)

Modélisation Orientée-Objet

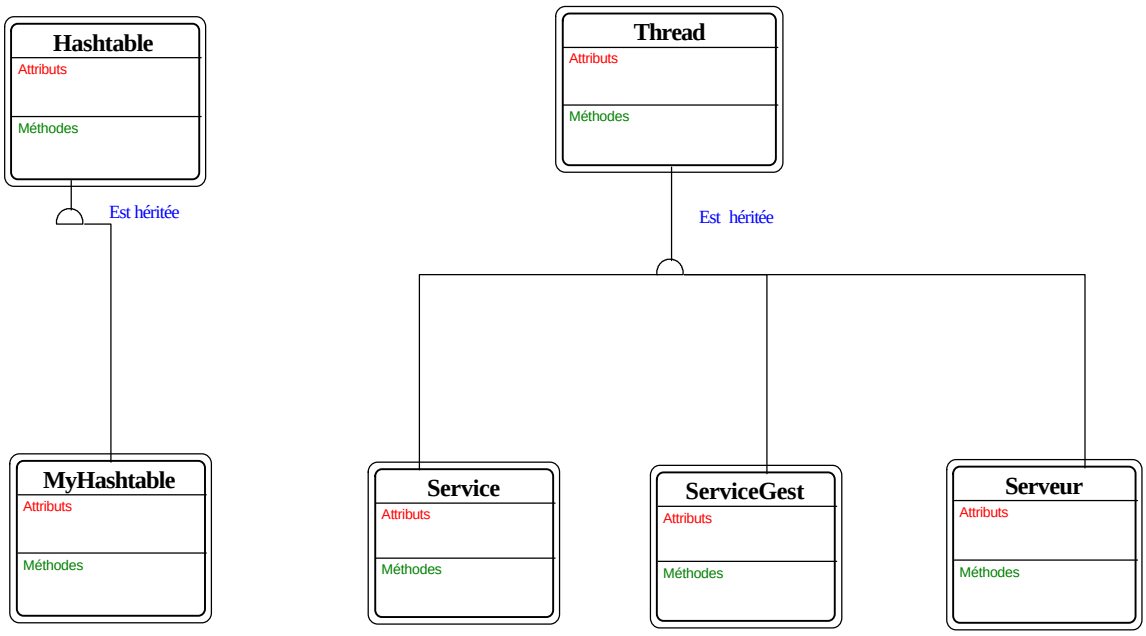


Figure 2.7: Le modèle Orienté-Objet (suite)

Notons que:

- Service, ServiceGest et Serveur **Sont hérités** du fil d'exécution (Thread).
- MyHashtable **Est héritée** de Hashtable.

L'annexe D décrit d'une façon détaillée toutes les classes avec leurs attributs et leurs méthodes.

2.9 Associations entre les classes de la base

Pour clarifier la modélisation objet de la base d'inspection des pièces et pour que cette modélisation soit compréhensible par les utilisateurs du système d'inspection automatisée dans les différents laboratoires, il s'avère préférable d'expliquer chaque relation existant entre deux classes d'objets de la modélisation.

1. Pièce **Fait partie** Base

En effet, une pièce est un membre de:

• plusieurs bases (bases réduites).

Et une base est une collection de:

• au minimum une pièce.

• au maximum plusieurs pièces.

2. Pièce **Peut avoir** Fichier

Une pièce peut avoir:

- au minimum deux fichiers: description CAO et aspect visuel .GIF.
- au maximum plusieurs fichiers.

Remarque: Deux pièces ne peuvent jamais avoir les mêmes fichiers, ni les mêmes versions.

3. Pièce **Peut avoir** Fichier2D

Une pièce peut avoir:

- au minimum aucun fichier 2D.
- au maximum plusieurs fichiers 2D.

4. Fichier **Posséder** Type

- Un fichier possède un et un seul type.
- Un type peut caractériser de 0 à N fichiers.

5. Fichier2D **Posséder** Type

- Un fichier2D possède un et un seul type.
- Un type peut caractériser de 0 à N fichiers2D.

6. Pièce **Posséder** Vue géon

- Une pièce a une et une seule vue géon.
- Une vue géon correspond à une et une seule pièce.

7. Vue géon **Avoir** Connexion

Une vue géon demande:

- au minimum aucune connexion, et au maximum une seule connexion entre un couple de deux géons.

Une connexion est un membre de:

- exactement une vue géon.

8. Vue géon **Avoir** Géon utilisé

Une vue géon peut avoir:

- au minimum un géon utilisé.
- au maximum plusieurs géons utilisés.

Un géon utilisé est un membre de:

- au minimum une vue géon.
- au maximum plusieurs vues géons. Par exemple, le géon 11 peut être dans les pièces 3, 5, 6, 9 et 10.

9. Connexion **Fait partie** Couple

- Un connexion correspond à un couple de 2 identificateurs (id1, id2).
- Chaque couple a au plus une connexion.

10. Géon **Fait partie** Géon utilisé

Un géon fait partie de:

- au minimum aucun géon utilisé.
- au maximum plusieurs géons utilisés.

11. Attribut **Fait partie** Connexion

Une connexion est un constitué de:

- au minimum un attribut.
- au maximum plusieurs attributs.

Un attribut est un composant de:

- au minimum aucune connexion.
- au maximum plusieurs connexions.

12. Géon **Est caractérisé** Attribut

Chaque géon identifié est caractérisé par:

- au minimum aucun attribut.
- au maximum plusieurs attributs (pour le moment 5 attributs).

13. Attribut **Avoir** Valeur

- Chaque attribut doit avoir une et une seule valeur.
- Une valeur correspond au minimum à un seul attribut et au maximum à plusieurs attributs. Par exemple la valeur d'attribut 1 correspond aux noms attributs: *Axe*, *Matérialité*, *Arêtes*, *Symétrie* et *Dimension* .

Le chapitre suivant traite les mécanismes de communication (Client-Serveur) à partir de la douille afin d'interroger la base de données localement et à distance.

CHAPITRE 3

ARCHITECTURE CLIENT-SERVEUR DE LA BASE DE DONNÉES

Nous nous intéressons dans ce chapitre à approfondir les concepts reliés à la communication entre un client et un serveur à partir d'une douille (*socket*). Ce chapitre introduit le protocole TCP/IP (*Transmission Control Protocol/Internet Protocol*), comme il clarifie l'architecture Client/Serveur utilisée. Il analyse le principe d'utilisation d'une douille. Enfin, il liste les requêtes utilisées à partir des clients pour questionner la base de données locale et à distance du serveur.

3.1 Les douilles d'UNIX

Le rôle d'une douille est semblable à un appareil de téléphone alors que le protocole TCP/IP est le langage utilisé pour la communication. Donc il est important de bien comprendre le fonctionnement de chacun de ces composants.

3.1.1 Définition

Les douilles sont des mécanismes d'interface de programmation qui permettent aux programmes d'échanger des données. Les douilles sont à la frontière entre l'espace de l'utilisateur et l'espace noyau d'UNIX.

Remarque:

Au niveau de *Windows NT*, la douille fonctionne de la même façon que UNIX sauf qu'elle fait son appel à *Win Sock.*

3.1.2 Introduction

Avec les protocoles TCP, une connexion est entièrement définie par les cinq paramètres suivants:

— le type du protocole TCP,
— l'adresse Internet de la première machine,
— le numéro du port associé au processus s'exécutant sur la première machine,
— l'adresse Internet de la seconde machine,
— le numéro du port associé au processus s'exécutant sur la seconde machine.

Le type du protocole étant commun aux deux applications, cela se traduit localement sur chaque machine, par trois valeurs:

— type du protocole TCP,
— adresse Internet,
— numéro du port associé au processus.

Ces trois informations doivent être fournies au système lors de la création d'une douille.

La connexion, du moins dans sa phase d'établissement, n'est pas symétrique. On appelle:
— serveur, une application qui attend les demandes d'ouverture de connexion,
— client, une application qui effectue ces demandes d'ouverture.

Sur un serveur devront être définis le protocole utilisé et le numéro du port local sur lequel les connexions devront être attendues. Pour le client, devront être définis le protocole, l'adresse de la machine distante et le numéro de port sur lequel elle attend les connexions. Les autres données seront déduites par le système. Ainsi le client et le serveur connaissent leur propre adresse. Le numéro de port du client sera alloué dynamiquement par le système d'exploitation en fonction des numéros de port déjà utilisés localement.

3.1.3 Les primitives TCP pour les douilles

Les primitives TCP pour les douilles permettent:

- L'attente passive de connexions TCP initiées par des applications distantes;
- L'initiation de connexions actives vers des applications distantes en attente passive;
- L'acceptation ou le refus d'une connexion initiée par une application distante;
- L'envoi et la réception de données sur une connexion établie;
- La terminaison brutale ou ordonnée d'une connexion établie;
- Le test d'une connexion.

3.1.4 Enchaînement des procédures

En théorie, l'utilisation des fonctions manipulant les douilles est aussi simple que la manipulation des fichiers sous UNIX. On retrouve d'ailleurs certaines commandes communes. L'apparente difficulté vient du fait qu'il faille enchaîner plusieurs primitives pour arriver à un résultat.

3.1.4.1 Mode connecté

La figure 3.1 ci-dessous, indique l'enchaînement des primitives pour le mode connecté.

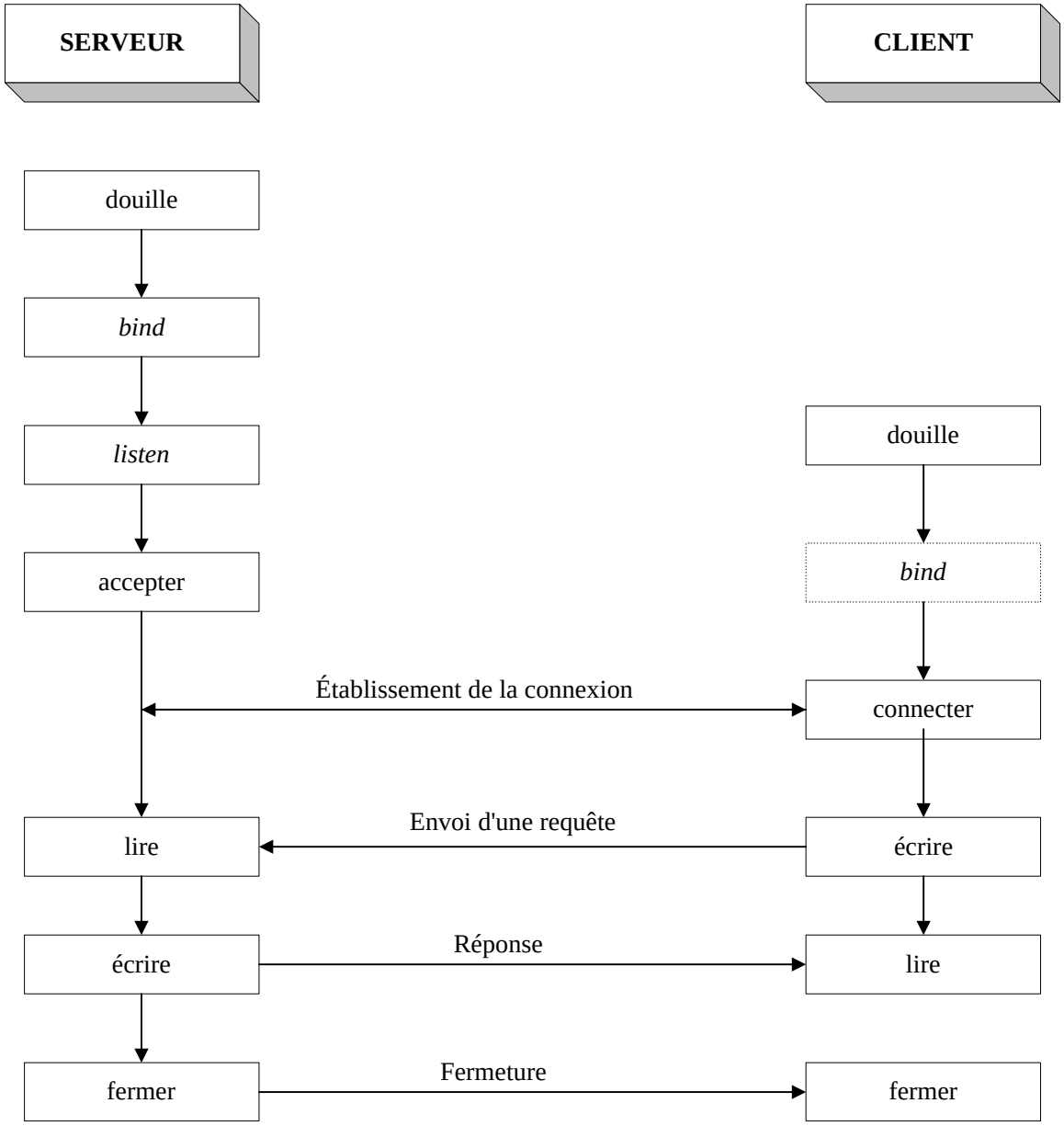


Figure 3.1: Enchaînement des primitives pour le mode connecté

Pour l'application serveur:

- l'appel système *douille* permet d'indiquer au système que l'application désire communiquer avec un autre programme. Cet appel système retourne un identificateur qui sera utilisé dans tous les autres appels système faisant référence à cette douille;

- l'appel système *bind* permet à l'application de fournir au noyau les paramètres nécessaires au fonctionnement de la douille. Pour un serveur, il s'agit du port sur lequel il attendra les demandes de connexion. Les douilles sont normalement utilisées pour échanger des données entre deux applications. Dans le cas d'une application serveur, les douilles servent à attendre les demandes de connexions;

- l'appel système *listen* permet de placer la douille et le protocole associé dans un état d'attente de requêtes de connexion;

- l'appel système *accepter* est bloquant. Il place le programme en attente effective de requête de connexion. Quand une connexion est établie par un client, l'appel système *accepter* se débloque et retourne un identificateur de douille sur lequel des données pourront être échangées;

- les appels système *lire* et *écrire* permettent l'échange de données. Il s'agit des mêmes instructions que pour les fichiers;

- l'appel système *fermer* permet de fermer localement la douille. S'il s'agit de la dernière référence à la connexion, cela se traduit par l'envoi sur le réseau de la séquence de fermeture.

Pour l'application client, la création de la douille est identique à celle du serveur:

- l'appel système *bind* est facultatif car, dans la plupart des cas, le numéro de port local identifiant la connexion peut être alloué dynamiquement par le système. Dans le cas contraire, l'appel système *bind* doit être utilisé comme pour le serveur;

- l'appel système *connecter* permet d'ouvrir une connexion avec une application serveur. Cette instruction va se traduire par l'envoi de la séquence d'ouverture sur le réseau. Si aucun message d'erreur n'est retourné, la connexion est établie.

À partir de l'ouverture de la connexion, le client et le serveur ont un comportement symétrique. Les instructions sont donc les mêmes [Toutain, 1996].

3.2 Le protocole TCP/IP

Le protocole TCP[1] (*Transmission Control Protocol*) est un protocole plus sûr bâti au dessus du protocole IP (*Internet Protocol*). Les données sont transmises sous forme de paquets. Le rôle de TCP est de sécuriser l'envoi et la réception des paquets par un système d'accusé de réception. C'est un protocole de type orienté connexion , c'est à dire que les échanges de données ont lieu de façon fiables, après négociation préliminaire des caractéristiques de transfert. En particulier, le protocole TCP garde de lui même une copie des données émises tant que celles-ci n'ont pas été reçues par le destinataire. En plus, le TCP garantit l'intégrité des données en ajoutant un code de vérification dans chaque paquet envoyé [Umar, 1993].

[1] Notre application de communication (Client/Serveur) via IP, il s'agit d'utiliser un protocole de transport fiable (TCP: *Transmission Control Protocol*).

3.2.1 TCP/IP et JAVA

Les sockets Java sont des mécanismes de communication de plus bas niveau que les URLs[2] bien que toujours indépendantes de la configuration matérielle et logicielle utilisée. Une douille est une liaison point à point entre un serveur et un client (le code des programme est donc légèrement différent). La communication est *full-duplex* (en fait, les communications sont probablement tamponnées avant d'être envoyées, d'où l'apparence du full duplex là où TCP/IP ne sait faire que du half-duplex) et le protocole d'échange des informations est laissée à la charge du programmeur [Umar, 1993].

3.2.2 Principe d'utilisation des douilles

Les douilles, mécanismes de communication entre processus, fonctionnent, que les processus soient sur la même machine ou non. Les processus échangent des données à travers un réseau, local ou distant.

Le principe d'utilisation des douilles est du type client-serveur: un processus déclare accepter les connexions des clients, et lorsqu'un client se présente, une douille est créée pour établir le lien entre les deux.

Le mécanisme des douilles n'est pas propre à Java, mais Java en fournit une implémentation portable, réalisée grâce aux classes: *ServerSocket* (qui permet de créer un serveur à l'écoute des connexions à venir) et *Socket* (qui permet d'initialiser la connexion avec un serveur) [Macary et Nicolas, 1996].

---

[2] Une URL désigne une ressource dans un réseau sous le format suivant: protocole://nom-hote[:port]/chemin/nom-fichier # section. En Java, l'URL n'est pas représenté par une chaîne de caractère mais par une instance de la classe **java.net.URL**.

3.2.2.1 Côté Client

Pour créer la douille du côté du programme client, le programmeur doit donner le nom de la machine où se trouve le serveur et le port où l'on doit se connecter, qui est un numéro arbitraire[3] décrivant le type de service:

ma_douille = new Socket(*hopfield.livia.etsmtl.ca* , 12000)

3.2.2.2 Côté Serveur

De son côté, le serveur doit créer un *serverSocket* en indiquant son numéro de port, puis attendre qu'un client demande une connexion qui peut alors être acceptée (cf. *Figure 3.2*). Des flux d'entrée/sortie sont ensuite définis et les échanges peuvent commencer selon le protocole choisi.

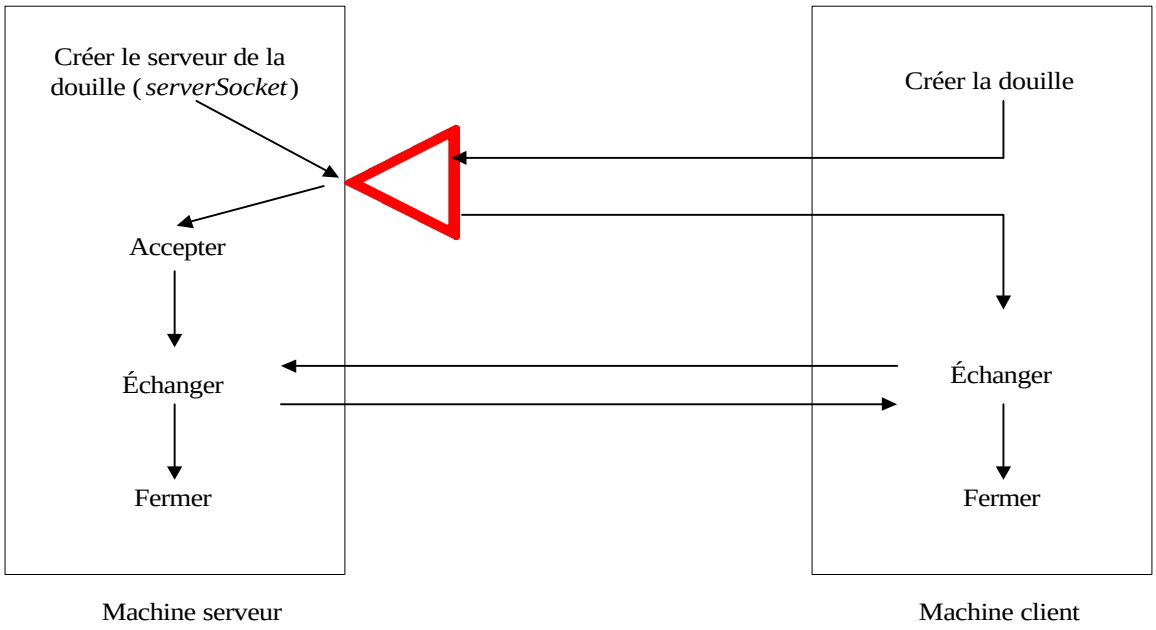


Figure 3.2: Scénario entre client et serveur

[3] Chaque application doit utiliser son propre numéro de port entre ceux qui sont disponibles. Ainsi, on ne peut pas partager le même port entre les différentes applications dans un seul serveur.

3.3 Architecture Client-Serveur

Le mode de communication qu'un hôte établit avec un autre hôte qui fournit un service quelconque est désigné par le terme Client/Serveur. Comme la base de données est partagée entre plusieurs laboratoires français et canadiens, il a fallu gérer les accès à la base par un système de Client/Serveur. Les notions de base de l'architecture client-Serveur sont:

- **Client**: Processus demandant l'exécution d'une opération à un autre processus par un envoi de message et attendant la réponse à cette question par un message en retour.
- **Serveur**: Processus accomplissant une opération sur demande d'un client et transmettant la réponse à ce client.
- **Requête**: Message transmis par un client à un serveur décrivant l'opération à exécuter pour le compte du client.
- **Réponse**: Message transmis par un serveur à un client suite à l'exécution d'une opération contenant les paramètres de retour de l'opération.

La figure 3.3 ci-dessous illustre l'architecture Client/Serveur:

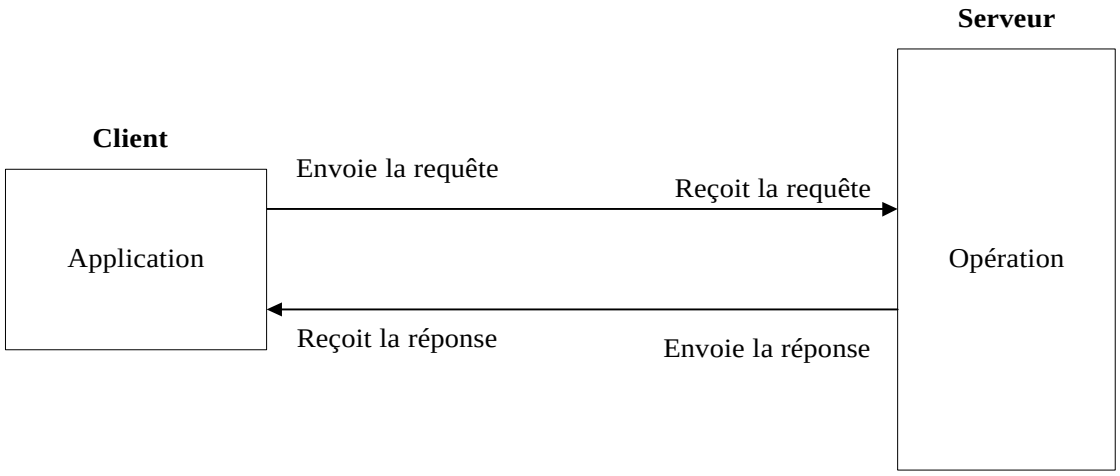


Figure 3.3: Dialogue Client-Serveur

3.3.1 La répartition Client-Serveur

Ici , on part du général des sections précédentes au cas particulier du projet.
Le niveau de communication entre le client et le serveur se fait par chaînes de caractères et par octets tel qu'il est illustré à la figure 3.4. En effet, un client envoie une requête sous forme d'une chaîne de caractères au serveur à partir de la douille. Le serveur reçoit la requête, la traite, et puis il retourne la réponse au client.

Lors du transfert ou téléchargement des fichiers, la communication Client/Serveur se fait octet par octet dans la même douille. Pratiquement le client est connecté avec le programme serveur qui se trouve sur *hopfield.livia.etsmtl.ca*. Il fait télécharger les fichiers pour les utiliser chez lui.

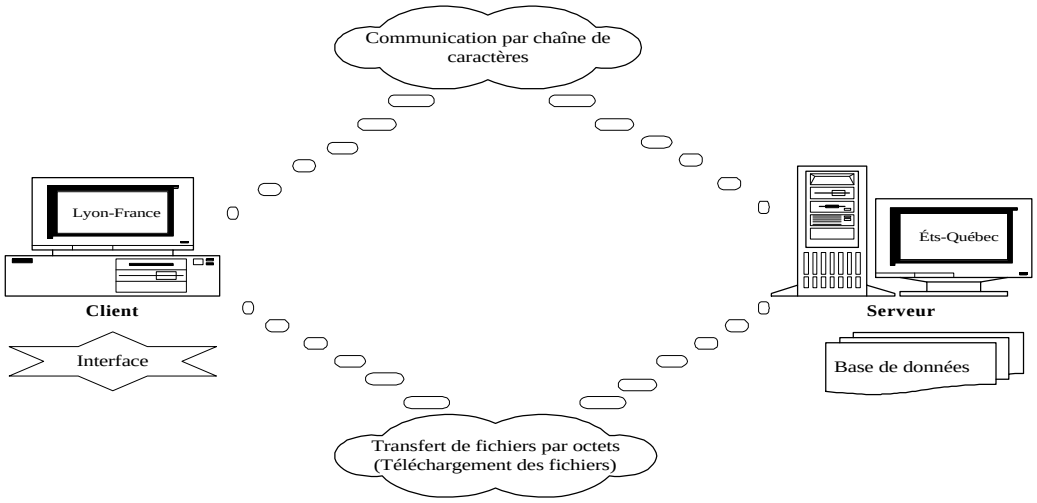


Figure 3.4: Répartition Client-Serveur

Remarque: Un client ne peut jamais accéder directement au serveur hôte[4]. Il peut accéder au canal (en mode de lecture ou bien d'écriture), de même pour le serveur (cf. figure 3.5 de la page suivante).

[4] Un réseau informatique est constitué d'un ensemble d'ordinateurs et de périphériques interconnectés. Un nœud du réseau qui correspond à un ordinateur est généralement appelé hôte.

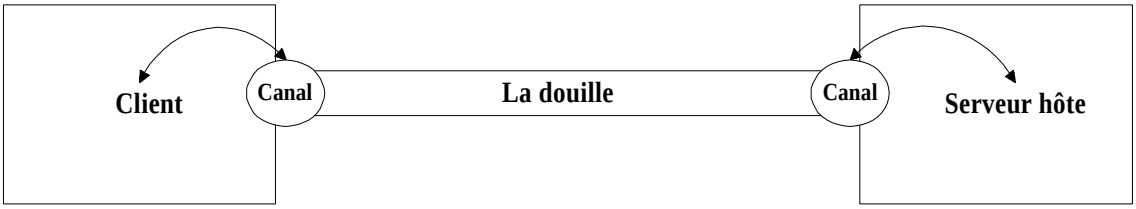


Figure 3.5: Client-Serveur et Canal

Le développement de deux logiciels (un pour la partie serveur et un pour la partie client) était donc nécessaire. La partie serveur est composée d'une base de données incluant toutes les informations sous forme de fichiers binaires. Par contre, la partie côté client est constituée d'un programme qui permet d'interroger[5] la base de données à distance. Les logiciels permettent aussi d'attacher[6] des fichiers à une pièce. Il a donc fallu prévoir le transfert de ces fichiers sous forme d'octets *(bytes)* d'un module à un autre.

3.4 Liste des requêtes acceptées par le serveur

Le client et le serveur peuvent alors communiquer l'un avec l'autre en écrivant et en lisant sur les douilles. Ainsi pour questionner la base, il suffit d'envoyer des requêtes au serveur (voir annexe E).

---

[5] Nous référons notre lecteur à l'annexe A qui illustre cet effet.

[6] Du côté serveur, on vous invite à voir la fenêtre Ajouter un fichier à une pièce dans l'annexe F, page 119, et du côté client, vous pouvez consulter l'annexe E.

Enfin, on peut dire qu'avec la douille JAVA et le protocole de communication TCP/IP, nous avons assuré un haut niveau d'interrogation qui est caractérisé par un court temps de réponse. Par exemple, quand un utilisateur distant veut télécharger chez lui des icônes de format .GIF de la base de données, nous avons remarqué que cette opération se fait presque instantanément.

3.5 Les fenêtres utilisées pour la base de données

L'annexe F traite des interfaces usager graphiques à partir des clients (utilisateurs) et des gestionnaires qui possèdent différents droits d'utilisation. Pour plus de détails, voir la matrice des flux entre acteurs (Annexe B), ainsi que le manuel d'utilisation et les droits d'accès aux programmes de la base de données d'inspection (Annexe C).

CHAPITRE 4

CONCLUSION ET RECOMMANDATION

Dans ce travail de recherche, nous avons été chargés de mettre au point, premièrement, une base de données Orientée-Objet (OO) sous forme de fichiers binaires en utilisant le langage ″JAVA″ et deuxièmement les outils permettant de l'interroger à distance dans le but d'une inspection automatisée de pièces manufacturées.

Au début, le choix d'une base de données était un problème difficile à résoudre. En effet, l'achat d'un tel logiciel est coûteux et doit être rentabilisé. Ainsi trois propositions étaient disponibles:

- la première était de créer la base de données en ″JAVA″ ou ″C++″ avec un système de requêtes pour questionner la base à distance. Ce système comporte un risque dans le sens où le temps d'attente d'une requête peut être plus long qu'avec un système de gestion de base de données (SGBD).
- la deuxième était de créer un SGBD Oracle: il s'agit d'un système relationnel souvent utilisé aussi bien dans un cadre académique que professionnel mais celui-ci reste très onéreux.
- la troisième était de créer un SGBD O_2: c'est un système de base de données OO. Ce dernier n'est pas encore très répandu dans les entreprises (ceci est dû à la nouveauté de ce type de SGBD), mais on peut le trouver dans certaines universités canadiennes (Laval, Alberta et au CRIM).

De façon générale, une base de données OO semble mieux correspondre à notre projet.

Nous avons remarqué qu'au laboratoire du LIVIA à l'ÉTS, le personnel travaille généralement en C, C++ ou JAVA sous Unix. D'ailleurs, il n'existe pas de SGBD assez performant permettant le développement de notre application. Suite à ces constatations, notre choix final s'est arrêté sur un développement utilisant le langage ″JAVA″ dans l'environnement de développement JDK de SUN.

En utilisant ″JAVA″, notre base de données est OO et ne dépend d'aucune plate-forme. Pour bien spécifier la modélisation de notre système, on a utilisé l'analyse Orientée-Objet (AOO) qui a permis de clarifier les divers problèmes rencontrés au cours de notre projet. Dans ce rapport, l'AOO a permis d'organiser l'analyse et la spécification en utilisant des méthodes d'organisation plus proches de la pensée des individus.

Pour optimiser la BD de façon plus rationnelle et plus efficace, nous avons pensé utiliser des fichiers en format binaire. Ces derniers sont plus difficiles à déchiffrer: il faut connaître la clé (index) pour avoir le contenu des données (voir ¶ 2.6.4, page 41). De plus, le format binaire est plus compact. Il s'agit là d'une caractéristique très importante pour ce genre de fichiers et comme notre BD est centralisée entre plusieurs laboratoires, il apparaît beaucoup mieux d'utiliser les fichiers binaires. En effet, les accès se feront ainsi plus rapidement.

En définitive, pour qu'un client autorisé interroge à distance notre BD, nous avons besoin d'un système de requêtes assez performant. Avec la douille JAVA, utilisant le protocole de communication TCP/IP, nous avons bien assuré un haut niveau d'interrogation qui est caractérisé par un court temps de réponse. Par exemple, quand un utilisateur distant veut télécharger chez lui des icônes de format .GIF de la BD, nous avons remarqué que cette opération se fait presque instantanément.

Avec ce téléchargement rapide et avec la facilité des accès à la BD depuis n'importe quel endroit, nous pouvons affirmer que nous avons réussi à atteindre des résultats similaires à d'autres types de BD (Oracle et O_2 par exemple) et ceci en utilisant la douille de JAVA et le protocole de communication TCP/IP.

Durant la réalisation du projet, le problème majeur réside dans le téléchargement des données dans un laps de temps très court.

Des améliorations sont possibles pour augmenter la performance de ce système. Une technologie prometteuse est le concept des *Agents-Mobiles (AM)*. Un agent mobile aura le rôle de lancer les requêtes à la BD. L'avantage des AM est la réduction de la bande passante nécessaire pour la communication. Par exemple s'il y a plusieurs accès à la BD, nous avons un risque de congestion dans le réseau informatique. Avec les AM, il n'est plus nécessaire de maintenir une communication constante avec la BD. Les AM sont autonomes, et peuvent exécuter les opérations sur la BD puis revenir à la source avec les informations désirées. Notons que, chez les AM, il existe des normes de sécurités comparables à des systèmes traditionnels.

En contrepartie, cette innovation technologique est très jeune. Nous avons évité de l'utiliser comme stratégie d'implantation dans notre projet. C'est une technologie récente et elle est encore au stade de développement et de validation. Son usage constitue un risque potentiel.

BIBLIOGRAPHIE

[1] Binford, T.0. (1971). «Visual perception by computer». *Proc. of IEEE Conf. on Systems Science and Cybernetics*, Miami, F.L.

[2] Biederman, I. (1985). Human image understanding: «recent research and a theory». *Computer Graphics, Vision, and Image Processing*, 32, pp. 29-73.

[3] Biederman, I. (1987). Recognition by components: «a theory of human image understanding». Psychological Review, 94, pp. 118-122.

[4] Dickinson, S., Pentland, A., Rosenfeld, A. (1992). 3D shape recovery using distributed aspect matching. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, 14(2), pp. 174-198.

[5] Grégoire, N. (1996). Étude des géons dans la reconnaissance de forme en vision artificielle, Rapport de projet synthèse.

[6] Lepage, R., Grégoire, N., Lavigne, D. (1998). Définition des géons et de leurs attributs, Rapport d'étape.

[7] KIMWon et Autres, (1991). A Distributed Object-Oriented Database System Supporting Shared And Private Databases. *ACM Transaction on Information Systems*, Vol.9, No 1, Janvier 1991, pp.31-51.

[8] Hurtubise, R. (1984). La gestion de l'information, Agence d'Arc Inc., Montréal, Québec.

[9] Coad, P., Yourdon, E. (1992). Analyse orientée objets. Masson, Paris & Prentice Hall, London.

[10] Rumbaugh, J. et al. (1996). OMT Modélisation et conception orientées objet. Masson, Paris & Prentice Hall, London.

[11] Taylor, D.A. (1993). Technologie orientée objet: Guide du décideur informatique. Addison-Wesley : France.

[12] Hess, D.A. (1993). «*A Primer of Object-Oriented Technology*». DATAPRO Managing Information Technology. Software 7685. 1-9.

[13] Jacobson, I., Christerson, M., Jonsson, P. et Overgaard, G. (1993). Le génie logiciel orienté objet: une approche fondée sur les cas d'utilisation. Addison-Wesley, France.

[14] Macary, J-F., Nicolas, C. (1996). Programmation Java. Eyrolles, Paris.

[15] Desfray, P. (1997). Modélisation par objets. Masson, Paris.

[16] Gunton, T. (1993). Technologies Des Systèmes D'information. Masson, Paris.

[17] Elmasri, N. (1989). Fundamentals of Database System. IEEE Computer Society Press, Los Alamitos, California.

[18] Folk, M.J., Zoellick, B. (1992). File Structure. Addisson-Wesley Publishing Company, Inc.

[19] Gabay, j. (1993). Apprendre et pratiquer Merise. Masson, Paris.

[20] Davis, A.M. (1993). Software requirements: objets, functions, and states. PTR Prentice Hall.

[21] Toutain, L. (1996). Réseaux locaux et Internet. Hermès, Paris.

[22] Umar, A. (1993). Distributed Computing and Client/Serveur Systems, Prentice Hall.

ANNEXE A

DIALOGUE CLIENT-SERVEUR

Interface graphique entre client et serveur

ANNEXE B

INVENTAIRE DES FLUX ET MATRICE DES FLUX DE DONNÉES

B1) Inventaire des flux:

- Pièce (Base de données).
- Pièce (Base de travail).
- Fichier 2D.
- Liste de géons existants.
- Géons d’une pièce.
- Connexion entre 2 géons dans une pièce.
- Liste attributs géon.
- Liste attributs connexion.
- Attributs d’une connexion.

B2) Matrice des flux des données:

Après avoir inventorié les flux entre acteurs, ceux-ci sont présentés ainsi:

De: / **Vers:**	**Client (utilisateur)**	**Gestionnaire de la Base de données**
Pièce (Base de données)	- Ajouter pièce - Télécharger images - Ajouter fichier CAO à la liste des fichiers - Mettre à jour fichier CAO - Supprimer fichier CAO - Ajouter fichier 2D - Supprimer fichier 2D - Télécharger images - Télécharger fichier CAO - Télécharger fichier 2D - Récupérer ancienne version 2D - Visualiser informations - Choisir répertoire	- Ajouter pièce - Supprimer pièce - Ajouter fichier CAO à la liste des fichiers - Mettre à jour fichier CAO - Supprimer fichier CAO - Ajouter fichier 2D - Supprimer fichier 2D - Télécharger images - Télécharger fichier CAO - Télécharger fichier 2D - Récupérer ancienne version 2D - Visualiser informations - Choisir répertoire
Pièce (Base de travail)	- Ajouter/supprimer base - Ajouter pièce - Supprimer pièce de la base de travail - Télécharger images - Spécifier répertoire local - Télécharger fichiers - Choisir répertoire - Visualiser - Limiter	- Ajouter/supprimer base - Ajouter pièce - Supprimer pièce de la base de travail - Télécharger images - Spécifier répertoire local - Télécharger fichiers - Choisir répertoire - Visualiser - Limiter

De: / Vers:	Client (utilisateur)	Gestionnaire de la Base de données
Fichier 2D	- Visualiser type fichier - Modifier type fichier - Visualiser type d'éclairage - Modifier type d'éclairage - Ajouter type - Mettre à jour type - Supprimer type - Télécharger - Récupérer ancienne version - Ajouter éclairage - Supprimer éclairage - Rechercher répertoire	- Visualiser type fichier - Modifier type fichier - Visualiser type d'éclairage - Modifier type d'éclairage - Ajouter type - Mettre à jour type - Supprimer type - Télécharger - Récupérer ancienne version - Ajouter éclairage - Supprimer éclairage - Rechercher répertoire
Liste de géons existants	- Visualiser - Détails/géon	- Ajouter géon - Supprimer géon - Visualiser - Détails/géon
Géons d'une pièce	- Ajouter géon - Supprimer géon - Détails/géon	- Ajouter géon - Supprimer géon - Détails/géon
Connexion entre 2 géons Dans une pièce	- Ajouter connexion - Supprimer connexion - Visualiser pièce - Détails/connexion	- Ajouter connexion - Supprimer connexion - Visualiser pièce - Détails/connexion
Liste attributs géon	- Visualiser	- Ajouter/Modifier attribut - Supprimer attribut - Ajouter/Modifier valeur - Supprimer valeur - Visualiser
Liste attributs connexion	- Visualiser	- Ajouter/Modifier attribut - Supprimer attribut - Ajouter/Modifier valeur - Supprimer valeur - Visualiser
Attributs d'une connexion	- Ajouter attribut - Modifier attribut - Supprimer attribut	- Ajouter attribut - Modifier attribut - Supprimer attribut

ANNEXE C

MANUEL D’UTILISATION

C.1) Programmes:

Il y a cinq programmes:

serveur: qui contient et qui gère la base de données.

client: interface graphique permettant la consultation et la mise à jour de la base par un utilisateur privilégié.

clientSI: programme sans interface permettant la consultation et la mise à jour de la base par un utilisateur privilégié.

clientGest: interface graphique permettant la consultation et la mise à jour de la base par un *gestionnaire.*

clientSIGest: programme sans interface permettant la consultation et la mise à jour de la base par un *gestionnaire.*

clientSI et clientSIGest attendent les commandes sur l'entrée standard et affichent les résultats sur la sortie standard (voir annexe A).

C2) Accès aux programmes:

- Pour appeler le programme ***²client²***, il faut taper la commande *client* dans la fenêtre (terminal) d'Unix sur une machine de LIVIA.

- Pour avoir un programme ***²client²*** fonctionnel sur une autre machine localisée en dehors du réseau (livia.etsmtl.ca), il faut:

1) transférer les exécutables ***²client²*** sur la machine externe et démarrer la commande *client*.
2) permettre l'accès par *telnet* sur les machines de LIVIA; cette approche nécessite que l'usager de l'extérieur a un compte actif sur ces machines.

La première approche est plus conforme à l'optique de développement d'une application Client-Serveur utilisant le port *http* (80).

- À la suite d'une interruption de service au LIVIA, pour activer le serveur de la base de données, il faut se connecter sur *hopfield* et exécuter la commande *serveur*. Çà doit être fait par quelqu'un qui est membre du groupe d'inspection.

- Pour se connecter en *client-Gestionnaire:* C'est le même processus que *client*, sauf que la commande est *clientGest.*

ANNEXE D

LE DICTIONNAIRE DES CLASSES

Cet annexe décrit les principales classes du modèle objet de la base de données des pièces. Chaque classe possède des attributs et des méthodes citées ci-dessous. Le symbole * devant un nom d'attribut indique une variable de classe.

Variable de classe: Chaque objet possède ses propres attributs. Il est possible que tous les objets d'une même classe aient des attributs en commun: ce sont les attributs de classe, introduits par le mot-clé *static*. La modification d'un attribut *static* est immédiatement prise en compte par les autres objets, puisqu'ils le partagent.

D.1 Classe *Base*

D.1.1 Description: Représente la base qui contient toutes les pièces.

D.1.2 Attributs

Nom	Description
* ListeBaseReduite	Liste des bases réduites.
ListePiece	Liste des pièces de la base.

D.1.3 Constructeur

D.1.3.1 Constructeur: Base()
Description: Crée la base.

D.1.4 Méthodes

D.1.4.1 Méthode: afficheListe()
Description: Permet de visualiser la liste des bases réduites sous forme de chaîne; utilisée pour construire les listes.

D.1.4.2 Méthode: ajouterBase(String nom, Vector tableau)
Description: Ajoute une nouvelle base à la liste des bases réduites.

D.1.4.3 Méthode: ajouterConnexion(int etiquettePiece, int id1, int id2, String nom, int valeur)
Description: Ajoute une connexion à une pièce de la base.

D.1.4.4 Méthode: ajouterFichier(int etiquettePiece, String nom, String type)
Description: Ajoute un fichier à une pièce de la base.

D.1.4.5 Méthode: ajouterFichier(int etiquettePiece, String nom, String type, String commentaire)
Description: Ajoute un fichier à une pièce de la base.

D.1.4.6 Méthode: ajouterFichier2D(int etiquettePiece, String nom, String type, Vector eclairage, int camera)
Description: Ajoute un fichier 2D à une pièce de la base.

D.1.4.7 Méthode: ajouterFichier2D(int etiquettePiece, String nom, String type, Vector eclairage, int camera, String commentaire)
Description: Ajoute un fichier 2D à une pièce de la base.

D.1.4.8 Méthode: ajouterGeon(int etiquettePiece, int etiquetteGeon)
Description: Ajoute un géon à une pièce de la base.

D.1.4.9 Méthode: ajouterNouveauGeon(int etiquette)
Description: Ajoute un nouveau géon à la liste des géons existants.

D.1.4.10 Méthode: ajouterNouvelAttributConnexion(String nom, int valeur, String signification)
Description: Ajoute une nouvelle valeur (et nouvel attribut si inexistant) à la liste des attributs connexions existants.

D.1.4.11 Méthode: ajouterNouvelAttributGeon(String nom, int valeur, String signification)
Description: Ajoute une nouvelle valeur (et nouvel attribut si inexistant) à la liste des attributs géons existants.

D.1.4.12 Méthode: ajouterPiece(Piece piece)
Description: Permet d'ajouter une pièce à la base de données.

D.1.4.13 Méthode: ajouterType(String nom, int unique)
Description: Permet d'ajouter un nouveau type à la liste des types de fichiers existants.

D.1.4.14 Méthode: ajouterType2D(String nom)
Description: Ajoute un nouveau type à la liste des types de fichiers 2D existants.

D.1.4.15 Méthode: ajouterTypeEclairage(String nom)
Description: Ajoute un nouveau type à la liste des types d'éclairage existants.

D.1.4.16 Méthode: ancienneVersionFichier(int etiquettePiece, String nom)
Description: Permet de revenir à l'ancienne version du fichier si possible.

D.1.4.17 Méthode: charger(int mode)
Description: Permet d'effectuer le chargement de la base à partir des fichiers binaires.

D.1.4.18 Méthode: commentaireFichier(int etiquettePiece, String nom)
Description: Récupère le commentaire d'un fichier associé à une pièce.

D.1.4.19 Méthode: commentaireFichier2D(int etiquettePiece, String nom)
Description: Récupère le commentaire d'un fichier 2D associé à une pièce.

D.1.4.20 Méthode: ContientGeon(int idGeon)
Description: Récupère la liste des pièces contenant le géon sous forme d'une chaîne de caractères.

D.1.4.21 Méthode: enregistrer(int mode)
Description: Enregistre la base dans des fichiers binaires.

D.1.4.22 **Méthode:** getListe()
Description: Récupère la liste des pièces.

D.1.4.23 **Méthode:** getListeBase()
Description: Récupère la liste des bases réduites.

D.1.4.24 **Méthode:** getListePiece()
Description: Récupère la liste des pièces de la base.

D.1.4.25 **Méthode:** getPiece(int etiquette)
Description: Récupère une pièce à partir de son étiquette.

D.1.4.26 **Méthode:** limiterBase(String nom)
Description: Limite la base que l'on va utiliser à la base réduite spécifiée.

D.1.4.27 **Méthode:** limiterBase(Vector arg)
Description: Limite la base que l'on va utiliser aux pièces passées en argument.

D.1.4.28 **Méthode:** ListeAttributConnexion(int etiquette, int id1, int id2)
Description: Récupère la liste des attributs de la connexion sous forme d'une chaîne de caractères.

D.1.4.29 **Méthode:** ListeAttributConnexionExistant()
Description: Récupère la liste des attributs connexions existants sous forme d'une chaîne de caractères.

D.1.4.30 **Méthode:** ListeAttributGeon(int etiGeon)
Description: Récupère la liste des attributs du géon sous forme d'une chaîne de caractères.

D.1.4.31 **Méthode:** ListeAttributGeonExistant()
Description: Récupère la liste des attributs géons existants sous forme d'une chaîne de caractères.

D.1.4.32 **Méthode:** ListeConnexion(int etiquette)
Description: Récupère la liste des connexions d'une pièce sous forme d'une chaîne de caractères.

D.1.4.33 **Méthode:** ListeFichier(int etiquette)
Description: Récupère la liste des fichiers sous forme d'une chaîne de caractères.

D.1.4.34 **Méthode:** ListeFichier2D(int etiquette)
Description: Récupère la liste des fichiers 2D sous forme d'une chaîne de caractères.

D.1.4.35 **Méthode:** ListeFichierGif()
Description: Récupère la liste des fichiers Gif de la base sous forme d'une chaîne de caractères.

D.1.4.36 **Méthode:** ListeFichierType(String type)
Description: Récupère la liste des fichiers du type spécifié de la base sous forme d'une chaîne de caractères.

D.1.4.37 **Méthode:** ListeGeon(int etiquette)
Description: Récupère la liste des géons d'une pièce sous forme d'une chaîne de caractères.

D.1.4.38 **Méthode:** ListeGeonExistant()
Description: Récupère la liste des géons existants sous forme d'une chaîne de caractères.

D.1.4.39 **Méthode:** ListePiece()
Description: Récupère la liste des pièces sous forme d'une chaîne de caractères.

D.1.4.40 **Méthode:** ListeTypeEclairage()
Description: Récupère la liste des types d'éclairage sous forme d'une chaîne de caractères.

D.1.4.41 **Méthode:** ListeTypeFichier()
Description: Récupère la liste des types des fichiers sous forme d'une chaîne de caractères.

D.1.4.42 **Méthode:** ListeTypeFichier2D()
Description: Récupère la liste des types des fichiers 2D sous forme d'une chaîne de caractères.

D.1.4.43 **Méthode:** ListeTypeFichier2DPlus()
Description: Récupère la liste des types des fichiers sous forme d'une chaîne de caractères avec le commentaire (Fichier 2D).

D.1.4.44 **Méthode:** ListeTypeFichierNonUnique()
Description: Récupère la liste des types des fichiers non unique sous forme d'une chaîne de caractères.

D.1.4.45 **Méthode:** maxGeon()
Description: Récupère la plus grande étiquette de géon existant +1.

D.1.4.46 **Méthode:** modifierAttributConnexion(int etiquettePiece, int id1, int id2, String nom, int valeur)
Description: Ajoute un attribut et sa valeur à une connexion ou modifie la valeur si l'attribut existe déjà dans la connexion.

D.1.4.47 **Méthode:** modifierAttributGeon(int idGeon, String nom, int valeur)
Description: Ajoute ou modifie la valeur d'un attribut à un géon de la liste des géons existants.

D.1.4.48 **Méthode:** modifierCommentaireFichier(int etiquettePiece, String nom, String comment)
Description: Modifie le commentaire d'un fichier d'une pièce de la base sans modifier sa version.

D.1.4.49 **Méthode:** modifierCommentaireFichier2D(int etiquettePiece, String nom, String comment)
Description: Modifie le commentaire d'un fichier 2D d'une pièce de la base sans modifier sa version.

D.1.4.50 **Méthode:** modifierFichier(int etiquettePiece, String nom, String comment)
Description: Modifie un fichier d'une pièce de la base, modifie la version et le commentaire.

D.1.4.51 Méthode: modifierFichier2D(int etiquettePiece, String nom, String comment)
Description: Modifie un fichier 2D d'une pièce de la base, modifie la version et le commentaire.

D.1.4.52 Méthode: NbContientGeon(int idGeon)
Description: Récupère le nombre de pièces contenant le géon sous forme d'une chaîne de caractères.

D.1.4.53 Méthode: remplissage()
Description: Crée un fichier *remplissage* qui contient tous les ordres nécessaires pour reconstruire la base de données.

D.1.4.54 Méthode: setListe(MyHashtable table)
Description: Définit la liste des pièces.

D.1.4.55 Méthode: setListeBase(MyHashtable liste)
Description: Met à jour la liste des bases réduites.

D.1.4.56 Méthode: setListePiece(MyHashtable liste)
Description: Définit la liste des pièces de la base.

D.1.4.57 Méthode: supprimerAttributConnexion(int etiquettePiece, int id1, int id2, String nom)
Description: Supprime un attribut et sa valeur d'une connexion.

D.1.4.58 Méthode: supprimerAttributGeon(int idGeon, String nom)
Description: Supprime un attribut d'un géon de la liste des géons existants.

D.1.4.59 Méthode: supprimerBase(String nom)
Description: Supprime la base réduite de la liste.

D.1.4.60 Méthode: supprimerConnexion(int etiquettePiece, int id1, int id2)
Description: Supprime une connexion à une pièce de la base.

D.1.4.61 Méthode: supprimerFichier(int etiquettePiece, String nom)
Description: Supprime un fichier à une pièce de la base.

D.1.4.62 Méthode: supprimerFichier2D(int etiquettePiece, String nom)
Description: Supprime un fichier 2D à une pièce de la base.

D.1.4.63 Méthode: supprimerGeon(int etiquettePiece, int idGeon)
Description: Supprime un géon à une pièce de la base.

D.1.4.64 Méthode: supprimerNouveauGeon(int etiquette)
Description: Supprime un géon à la liste des géons existants.

D.1.4.65 Méthode: supprimerNouvelAttributConnexion(String nom, int valeur)
Description: Supprime une valeur (et attribut si plus de valeur) de la liste des attributs connexions existants.

D.1.4.66 Méthode: supprimerNouvelAttributGeon(String nom, int valeur)
Description: Supprime une valeur (et un attribut si plus de valeur) de la liste des attributs géons existants.

D.1.4.67 Méthode: supprimerPiece(int etiquette)
Description: Supprime la pièce d'étiquette *etiquette* de la base de données.

D.1.4.68 Méthode: supprimerPieceGest(int etiquette)
Description: Supprime la pièce d'étiquette *etiquette* de la base de données et efface les fichiers du disque.

D.1.4.69 Méthode: supprimerType(String nom)
Description: Supprime un type de la liste des types de fichiers existants sauf types uniques.

D.1.4.70 Méthode: supprimerType2D(String nom)
Description: Supprime un type de la liste des types de fichiers 2D existants.

D.1.4.71 Méthode: supprimerTypeEclairage(String nom)
Description: Supprime un type de la liste des types d'éclairage existants.

D.1.4.72 Méthode: supprimerTypeGest(String nom)
Description: Supprime un type de la liste des types de fichiers existants.

D.1.4.73 Méthode: toString()
Description: Permet de visualiser la base sous forme de chaîne; utilisée pour construire les listes.

D.2 Classe *Piece*

D.2.1 Description: Décrit une pièce de la base.

D.2.2 Attributs

Nom	Description
Commentaire	Commentaire de la pièce.
Etiquette	Étiquette de la pièce.
ListeFichier	Liste des fichiers de la pièce.
ListeFichier2D	Liste des fichiers 2D de la pièce.
ListeFichierUnique	Pile des fichiers uniques de la pièce.
* NbPiece	Nombre de pièces déjà crées.
VueGeon	VueGeon associée à la pièce.

D.2.3 Constructeurs

D.2.3.1 Méthode: Piece(String commentaire, String fichierGif, String fichier, String type)
Description: Crée une pièce.

D.2.3.2 Méthode: Piece(String commentPiece, String fichierGif, String comment1, String fichier, String type, String comment2)
Description: Crée une pièce.

D.2.4 Méthodes

D.2.4.1 Méthode: ajouterConnexion(int id1, int id2, String nom, int valeur)
Description: Ajoute une connexion à une pièce de la base.

D.2.4.2 Méthode: ajouterFichier(Fichier fichier)
Description: Ajoute un fichier à la pièce.

D.2.4.3 Méthode: ajouterFichier(String nom, String type)
Description: Ajoute un fichier à la pièce.

D.2.4.4 Méthode: ajouterFichier(String nom, String type, String commentaire)
Description: Ajoute un fichier à la pièce.

D.2.4.5 Méthode: ajouterFichier2D(Fichier2D fichier)
Description: Ajoute un fichier 2D à la pièce.

D.2.4.6 Méthode: ajouterFichier2D(String nom, String type, Vector eclairage, int camera)
Description: Ajoute un fichier 2D à la pièce.

D.2.4.7 Méthode: ajouterFichier2D(String nom, String type, Vector eclairage**,** int camera**,** String commentaire)
Description: Ajoute un fichier 2D à la pièce.

D.2.4.8 Méthode: ajouterGeon(int etiquette)
Description: Ajoute un géon à la pièce.

D.2.4.9 Méthode: ancienneVersionFichier(String nom)
Description: Revient à l'ancienne version du fichier si possible.

D.2.4.10 Méthode: commentaireFichier(String nom)
Description: Récupère le commentaire d'un fichier.

D.2.4.11 Méthode: commentaireFichier2D(String nom)
Description: Récupère le commentaire d'un fichier 2D.

D.2.4.12 Méthode: getCommentaire()
Description: Récupère le commentaire associé à une pièce.

D.2.4.13 Méthode: getEtiquette()
Description: Récupère l'étiquette de la pièce.

D.2.4.14 Méthode: getFichier(String nom)
Description: Récupère un fichier dans la liste des fichiers d'une pièce.

D.2.4.15 Méthode: getFichier2D(String nom)
Description: Récupère un fichier 2D dans la liste des fichiers d'une pièce.

D.2.4.16 Méthode: getListeFichier()
Description: Récupère la liste des fichiers de la pièce.

D.2.4.17 Méthode: getListeFichier2D()
Description: Récupère la liste des fichiers 2D de la pièce.

D.2.4.18 Méthode: getListeFichierUnique()
Description: Récupère la pile des fichiers uniques associée à une pièce.

D.2.4.19 Méthode: getNbPiece()
Description: Récupère le nombre de pièces déjà créées.

D.2.4.20 Méthode: getVueGeon()
Description: Récupère la vue géon de la pièce.

D.2.4.21 Méthode: listeFichier()
Description: Permet de visualiser la liste des fichiers sous forme de chaîne; utilisée pour construire les listes.

D.2.4.22 Méthode: listeFichier2D()
Description: Permet de visualiser la liste des fichiers 2D sous forme de chaîne; utilisée pour construire les listes.

D.2.4.23 Méthode: modifierAttributConnexion(int id1, int id2, String nom, int valeur)
Description: Ajoute un attribut et sa valeur à une connexion ou modifie la valeur si l'attribut existe déjà dans la connexion.

D.2.4.24 Méthode: modifierCommentaireFichier(String nom, String comment)
Description: Modifie le commentaire d'un fichier de la liste.

D.2.4.25 Méthode: modifierCommentaireFichier2D(String nom, String comment)
Description: Modifie le commentaire d'un fichier 2D de la liste.

D.2.4.26 Méthode: modifierFichier(String nom, String comment)
Description: Modifie le commentaire et la version d'un fichier de la liste.

D.2.4.27 Méthode: modifierFichier2D(String nom, String comment)
Description: Modifie le commentaire et la version d'un fichier 2D de la liste.

D.2.4.28 Méthode: setNbPiece(int nb)
Description: Met à jour le nombre des pièces créées.

D.2.4.29 Méthode: supprimerAttributConnexion(int id1, int id2, String nom)
Description: Supprime un attribut et sa valeur d'une connexion.

D.2.4.30 Méthode: supprimerConnexion(int id1, int id2)
Description: Supprime un connexion à une pièce de la base.

D.2.4.31 Méthode: supprimerFichier(String nom)
Description: Supprime un fichier de la liste des fichiers de la pièce.

D.2.4.32 Méthode: supprimerFichier2D(String nom)
Description: Supprime un fichier 2D de la liste des fichiers 2D de la pièce.

D.2.4.33 Méthode: supprimerGeon(int id)
Description: Supprime un géon à une pièce de la base.

D.2.4.34 Méthode: toString()
Description: Permet de visualiser une pièce sous forme de chaîne; utilisée pour construire les listes.

D.3 Classe *Fichier*

D.3.1 Description: Représente un fichier d'une pièce.

D.3.2 Attributs

Nom	Description
Commentaire	Commentaire du fichier.
* ListeTypeFichier	Liste des types de fichiers existants.
Nom	Nom du fichier avec extension.
Type	Type du fichier (extension).
Version	Version du fichier.

D.3.3 Constructeurs

D.3.3.1 Constructeur: Fichier(String nom, String type)
Description: Construit un fichier à partir d'un nom et d'un type.

D.3.3.2 Constructeur: Fichier(String nom, String type, String commentaire)
Description: Construit un fichier à partir d'un nom, d'un type et d'un commentaire.

D.3.4 Méthodes

D.3.4.1 Méthode: afficheListe()
Description: Permet de visualiser la liste des types de fichier existants sous forme de chaîne; utilisée pour construire les listes.

D.3.4.2 Méthode: afficheListeNonUnique()
Description: Permet de visualiser la liste des types de fichier non uniques sous forme de chaîne; utilisée pour construire les listes.

D.3.4.3 Méthode: ajouterType(Type type)
Description: Ajoute un nouveau type à la liste complète des types de fichier.

D.3.4.4 Méthode: ancienneVersion()
Description: Modifie la version (-1).

D.3.4.5 Méthode: contientType(String nomType)
Description: Pour savoir si la liste des types de fichier contient le type demandé.

D.3.4.6 Méthode: getCommentaire()
Description: Récupère le commentaire associé au fichier.

D.3.4.7 Méthode: getListe()
Description: Récupère la liste des types de fichiers existants.

D.3.4.8 Méthode: getNom()
Description: Récupère le nom du fichier.

D.3.4.9 Méthode: getType()
Description: Récupère le type du fichier.

D.3.4.10 Méthode: getVersion()
Description: Récupère la version du fichier.

D.3.4.11 Méthode: modif(String comment)
Description: Modifie la version (+1) et le commentaire du fichier.

D.3.4.12 Méthode: setCommentaire(String commentaire)
Description: Met à jour le commentaire du fichier.

D.3.4.13 Méthode: setListe(MyHashtable liste)
Description: Met à jour la liste des types de fichier existants.

D.3.4.14 Méthode: supprimerType(String nomType)
Description: Supprime le type de la liste complète des types de fichier, mais elle ne peut pas supprimer un type protégé (.gif, .iges et .neu).

D.3.4.15 Méthode: supprimerTypeGest(String nomType)
Description: Supprime le type de la liste complète des types de fichier.

D.3.4.16 Méthode: toString(String numPiece)
Description: Permet de visualiser un fichier sous forme de chaîne; utilisée pour construire les listes.

D.4 Classe *Fichier2D*

D.4.1 Description: Représente le fichier dimensionnel (2D) d'une pièce.

D.4.2 Attributs

Nom	Description
Camera	Angle de la caméra.
Commentaire	Commentaire du fichier.
Eclairage	Angles d'éclairage.
* ListeTypeEclairage	Liste des types d'éclairage existants.
* ListeTypeFichier	Liste des types de fichier 2D existants.
Nom	Nom du fichier avec extension.
Type	Type du fichier (extension).
Version	Version du fichier.

D.4.3 Constructeurs

D.4.3.1 Constructeur: Fichier2D(String nom, String type, Vector eclairage, int camera)
Description: Construit un fichier 2D à partir d'un nom, d'un type de fichier, des angles d'éclairage et d'un angle de caméra.

D.4.3.2 Constructeur: Fichier2D(String nom, String type, Vector eclairage, int camera, String commentaire)
Description: Construit un fichier 2D à partir d'un nom, d'un type de fichier, des angles d'éclairage, d'un angle de caméra et d'un commentaire.

D.4.4 Méthodes

D.4.4.1 Méthode: afficheListe()
Description: Permet de visualiser la liste des types de fichier existants sous forme de chaîne; utilisée pour construire les listes.

D.4.4.2 Méthode: afficheListe2D()
Description: Permet de visualiser la liste des types de fichier existants sous forme de chaîne avec le commentaire (Fichier 2D); utilisée pour construire les listes.

D.4.4.3 Méthode: afficheListeEclairage()
Description: Permet de visualiser la liste des types d'éclairage existants sous forme de chaîne avec le commentaire (Fichier 2D); utilisée pour construire les listes.

D.4.4.4 Méthode: ajouterType(Type type)
Description: Ajoute un nouveau type à la liste complète des types de fichier.

D.4.4.5 Méthode: ajouterTypeEclairage(String Type)
Description: Ajoute un nouveau type à la liste complète des types d'éclairage.

D.4.4.6 Méthode: ancienneVersion()
Description: Modifie la version (-1).

D.4.4.7 Méthode: contientType(String nomType)
Description: Pour savoir si la liste des types de fichier contient le type demandé.

D.4.4.8 Méthode: getCamera()
Description: Récupère l'angle de la caméra du fichier.

D.4.4.9 Méthode: getCommentaire()
Description: Récupère le commentaire associé au fichier.

D.4.4.10 Méthode: getEclairage()
Description: Récupère les angles d'éclairage du fichier.

D.4.4.11 Méthode: getListe()
Description: Récupère la liste des types de fichier existants.

D.4.4.12 Méthode: getListeEclairage()
Description: Récupère la liste des types d'éclairage existants.

D.4.4.13 Méthode: getNom()
Description: Récupère le nom du fichier.

D.4.4.14 Méthode: getType()
Description: Récupère le type du fichier.

D.4.4.15 Méthode: getVersion()
Description: Récupère la version du fichier.

D.4.4.16 Méthode: modif(String comment)
Description: Modifie la version (+1) et le commentaire du fichier.

D.4.4.17 Méthode: setCommentaire(String commentaire)
Description: Met à jour le commentaire du fichier.

D.4.4.18 Méthode: setListe(MyHashtable liste)
Description: Met à jour la liste des types de fichier existants.

D.4.4.19 Méthode: setListeEclairage(MyHashtable liste)
Description: Met à jour la liste des types d'éclairage existants.

D.4.4.20 Méthode: supprimerType(String nomType)
Description: Supprime le type de la liste complète des types de fichier, mais ne peut pas supprimer un type unique.

D.4.4.21 Méthode: supprimerTypeEclairage(String nomType)
Description: Supprime le type de la liste complète des types d'éclairage.

D.4.4.22 Méthode: supprimerTypeGest(String nomType)
Description: Supprime le type de la liste complète des types de fichier.

D.4.4.23 Méthode: toString(String numPiece)
Description: Permet de visualiser un fichier sous forme de chaîne; utilisée pour construire les listes.

D.5 Classe *Eclairage*

D.5.1 Description: Décrit le détail d'un éclairage pour un fichier 2D.

D.5.2 Attributs

Nom	Description
Angle	Angle de l'éclairage.
Puissance	Puissance de l'éclairage en *Watts.*
Type	Type de l'éclairage.

D.5.3 Constructeur

D.5.3.1 Constructeur: Eclairage(String angle, String type, String puissance)
Description: Construit un éclairage pour un fichier 2D.

D.5.4 Méthode

D.5.4.1 Méthode: toString()
Description: Permet de visualiser un éclairage sous forme de chaîne.

D.6 Classe *Type*

D.6.1 Description: Décrit le type de fichier.

D.6.2 Attributs

Nom	Description
NomType	Nom du type.
Unique	Booléen pour savoir si une pièce peut contenir plusieurs fichiers de ce type ou non.

D.6.3 Constructeur

D.6.3.1 Constructeur: Type(String nomType, boolean unique)
Description: Construit un type à partir d'un nom et d'un booléen.

D.6.4 Méthodes

D.6.4.1 Méthode: getNomType()
Description: Récupère le nom du type.

D.6.4.2 Méthode: isUnique()
Description: Récupère le booléen unique.

D.6.4.3 Méthode: toString()
Description: Permet de visualiser un type sous forme de chaîne; utilisée pour construire les listes.

D.7 Classe *VueGeon*

D.7.1 Description: Décrit la vue géon d'une pièce.

D.7.2 Attributs

Nom	Description
IdGeon	Nombre de géons crées dans la pièce.
* ListeCompleteGeon	Liste complète de tous les géons existants.
ListeConnexion	Liste des connexions d'une pièce.
ListeGeon	Liste des géons d'une pièce.

D.7.3 Constructeur

D.7.3.1 Constructeur: VueGeon()
Description: Crée une vueGeon.

D.7.4 Méthodes

D.7.4.1 Méthode: afficheListe()
Description: Permet de visualiser la liste complète des géons existants sous forme de chaîne; utilisée pour construire les listes.

D.7.4.2 Méthode: ajouterConnexion(Connexion connexion)
Description: Ajoute une connexion à la liste des connexions de la pièce.

D.7.4.3 Méthode: ajouterConnexion(int id1, int id2, String nom, int valeur)
Description: Ajoute une connexion à la liste des connexions de la pièce.

D.7.4.4 Méthode: ajouterGeon(int etiquette)
Description: Ajouter un géon à la liste des géons de la pièce.

D.7.4.5 Méthode: ajouterValeurGeon(Geon geon)
Description: Ajouter un nouveau géon à la liste des géons existants.

D.7.4.6 Méthode: contientGeon(int etiquette)
Description: Récupère le nombre de géon ayant comme étiquette e*tiquette* dans la liste des géons d'une pièce.

D.7.4.7 Méthode: getConnexion(int id1, int id2)
Description: Récupère une connexion.

D.7.4.8 Méthode: getConnexion(String cle)
Description: Récupère une connexion.

D.7.4.9 Méthode: getGeon(int id)
Description: Récupère un géon de la liste des géons existants.

D.7.4.10 **Méthode:** getGeonUtilise(int id)
Description: Récupère un géon utilisé d'après son identificateur.

D.7.4.11 **Méthode:** getListe()
Description: Récupère la liste complète des géons existants.

D.7.4.12 **Méthode:** getListeConnexion()
Description: Récupère la liste des connexions de la pièce.

D.7.4.13 **Méthode:** getListeGeon()
Description: Récupère la liste des géons de la pièce.

D.7.4.14 **Méthode:** maxGeon()
Description: Récupère la plus grande étiquette des géons de la liste des géons existants +1.

D.7.4.15 **Méthode:** modifierAttributConnexion(int id1, int id2, String nom, int valeur)
Description: Ajoute un attribut et sa valeur à une connexion ou modifie la valeur si l'attribut existe déjà dans la connexion.

D.7.4.16 **Méthode:** modifierAttributGeon(int idGeon, String nom, int valeur)
Description: Ajoute ou modifie la valeur d'un attribut à un géon de la liste des géons existants.

D.7.4.17 **Méthode:** setListe(MyHashtable liste)
Description: Définit la liste complète des géons existants.

D.7.4.18 **Méthode:** supprimerAttributConnexion(int id1, int id2, String nom)
Description: Supprime un attribut et sa valeur d'une connexion.

D.7.4.19 **Méthode:** supprimerAttributGeon(int idGeon, String nom)
Description: Supprime un attribut d'un géon de la liste des géons existants.

D.7.4.20 **Méthode:** supprimerConnexion(Couple couple)
Description: Supprime la connexion de la liste des connexions de la pièce.

D.7.4.21 **Méthode:** supprimerConnexion(int id1, int id2)
Description: Supprime la connexion de la liste des connexions de la pièce.

D.7.4.22 **Méthode:** supprimerGeon(int id)
Description: Supprime un géon de la liste des géons d'une pièce.

D.7.4.23 **Méthode:** supprimerValeurGeon(int etiquette)
Description: Supprime un géon de la liste des géons existants.

D.7.4.24 **Méthode:** toStringConnexion()
Description: Permet de visualiser la liste des connexions sous forme de chaîne; utilisée pour construire les listes.

D.7.4.25 **Méthode:** toStringGeon()
Description: Permet de visualiser la liste des géons sous forme de chaîne; utilisée pour construire les listes.

D.8 Classe *GeonUtilise*

D.8.1 Description: Représente le géon d'une pièce avec un identificateur.

D.8.2 Attributs

Nom	Description
Geon	Géon associé au géon utilisé.
Identifiant	Identifiant du géon utilisé.

D.8.3 Constructeur

D.8.3.1 Constructeur: GeonUtilise(int identifiant, Geon geon)
Description: Construit un géon utilisé.

D.8.4 Méthodes

D.8.4.1 Méthode: getGeon()
Description: Récupère le géon du géon utilisé.

D.8.4.2 Méthode: getIdentifiant()
Description: Récupère l'identifiant du géon utilisé.

D.8.4.3 Méthode: toString()
Description: Permet de visualiser un géon utilisé sous forme de chaîne; utilisée pour construire les listes.

D.8.4.4 Méthode: toSimpleString()
Description: Identique à *toString*, mais de manière moins complète utilisée pour construire les listes.

D.9 Classe *Geon*

D.9.1 Description: Modélise les géons d'une pièce.

D.9.2 Attributs

Nom	Description
Etiquette	Etiquette du géon.
ListeAttribut	Liste des attributs du géon.
* ListeCompleteAttribut	Liste des attributs existants pour les géons.

D.9.3 Constructeur

D.9.3.1 Constructeur: Geon(int etiquette)
Description: Construit un géon.

D.9.4 Méthodes

D.9.4.1 Méthode: afficheListe()
Description: Permet de visualiser la liste des attributs géons existants sous forme de chaîne; utilisée pour construire les listes.

D.9.4.2 Méthode: ajouterValeurAttribut(Attribut attribut)
Description: Ajoute une nouvelle valeur (et nouvel attribut si inexistant) à la liste des attributs géons existants.

D.9.4.3 Méthode: equals(Geon geon)
Description: Permet de comparer deux géons.

D.9.4.4 Méthode: getAttribut(String nom)
Description: Récupère un attribut de la liste des attributs du géon.

D.9.4.5 Méthode: getEtiquette()
Description: Récupère l'étiquette du géon.

D.9.4.6 Méthode: getListe()
Description: Récupère la liste complète des attributs géons existants.

D.9.4.7 Méthode: getListeAttribut()
Description: Récupère la liste des attributs du géon.

D.9.4.8 Méthode: modifierAttribut(String nom, int valeur)
Description: Ajoute un attribut et sa valeur à un géon ou modifie la valeur si l'attribut existe déjà dans la connexion.

D.9.4.9 Méthode: setListe(MyHashtable liste)
Description: Définit la liste complète des attributs géons existants.

D.9.4.10 Méthode: supprimerAttribut(String nom)
Description: Supprime un attribut et sa valeur d'un géon.

D.9.4.11 Méthode: supprimerValeurAttribut(String nom, int valeur)
Description: Supprime une valeur (et attribut si plus de valeur) de la liste des attributs géons existants.

D.9.4.12 Méthode: toString()
Description: Permet de visualiser un géon sous forme de chaîne; utilisée pour construire les listes.

D.9.4.13 Méthode: toStringSimple()
Description: Identique à *toString*, mais de manière moins complète utilisée pour construire les listes.

D.10 Classe *Attribut*

D.10.1 Description: Représente les attributs d'une connexion ou d'un géon.

D.10.2 Attributs

Nom	Description
Nom	Nom de l'attribut.
Valeur	Valeur de l'attribut.

D.10.3 Constructeurs

D.10.3.1 Constructeur: Attribut(String nom, int entier, String signification)
Description: Crée un attribut et sa valeur associée.

D.10.3.2 Constructeur: Attribut(String nom, Valeur valeur)
Description: Crée un attribut à partir d'une instance de valeur existante.

D.10.4 Méthodes

D.10.4.1 Méthode: equals(Attribut attribut)
Description: Permet de comparer deux attributs.

D.10.4.2 Méthode: getNom()
Description: Permet d'obtenir le nom de l'attribut.

D.10.4.3 Méthode: getValeur()
Description: Permet d'obtenir la valeur associée à l'attribut.

D.10.4.4 Méthode: toString()
Description: Permet de visualiser un attribut (Nom ou Valeur) sous forme de chaîne; utilisée pour construire les listes.

D.10.4.5 Méthode: toStringSimple()
Description: Identique à *toString*, mais de manière moins complète utilisée pour construire les listes.

D.11 Classe *Valeur*

D.11.1 Description: Décrit les valeurs d'un attribut d'un géon.

D.11.2 Attributs

Nom	Description
Entier	Valeur entière de la valeur.
Signification	Signification de la valeur.

D.11.3 Constructeur

D.11.3.1 Constructeur: Valeur(int entier, String signification)
Description: Crée une valeur.

D.11.4 Méthodes

D.11.4.1 Méthode: equals(Valeur valeur)
Description: Permet de comparer deux valeurs.

D.11.4.2 Méthode: getEntier()
Description: Récupère la valeur entière.

D.11.4.3 Méthode: getSignification()
Description: Récupère la signification de la valeur.

D.11.4.4 Méthode: toString()
Description: Permet de visualiser une valeur sous forme de chaîne; utilisée pour construire les listes.

D.11.4.5 Méthode: toStringSimple()
Description: Identique à *toString*, mais de manière moins complète utilisée pour construire les listes.

D.12 Classe *Connexion*

D.12.1 Description: Représente la connexion entre deux géons d'une pièce.

D.12.2 Attributs

Nom	Description
Couple	Couple de la connexion.
ListeAttribut	Liste des attributs de la connexion.
* ListeCompleteAttribut	Liste complète des attributs existants pour les connexions.

D.12.3 Constructeurs

D.12.3.1 Constructeur: Connexion(Couple couple, String nom, int valeur)
Description: Construit une connexion à partir d'un couple existant.

D.12.3.2 Constructeur: Connexion(int id1, int id2, String nom, int valeur)
Description: Construit une connexion et son couple.

D.12.4 Méthodes

D.12.4.1 Méthode: afficheListe()
Description: Permet de visualiser la liste des attributs connexions existants sous forme de chaîne; utilisée pour construire les listes.

D.12.4.2 Méthode: ajouterValeurAttribut(Attribut attribut)
Description: Ajoute une nouvelle valeur (et nouvel attribut si inexistant) à la liste des attributs connexions existants.

D.12.4.3 Méthode: equals(Connexion connexion)
Description: Permet de comparer deux connexions.

D.12.4.4 Méthode: getAttribut(String nom)
Description: Récupère un attribut de la liste des attributs de la connexion.

D.12.4.5 Méthode: getCouple()
Description: Récupère le couple de la connexion.

D.12.4.6 Méthode: getListe()
Description: Récupère la liste complète des attributs connexions existants.

D.12.4.7 Méthode: getListeAttribut()
Description: Récupère la liste des attributs de la connexion.

D.12.4.8 Méthode: modifierAttribut(String nom, int valeur)
Description: Ajoute un attribut et sa valeur à une connexion ou modifie la valeur si l'attribut existe déjà dans la connexion.

D.12.4.9 Méthode: setListe(MyHashtable liste)
Description: Définit la liste complète des attributs connexions existants.

D.12.4.10 Méthode: supprimerAttribut(String nom)
Description: Supprime un attribut et sa valeur d'une connexion.

D.12.4.11 Méthode: supprimerValeurAttribut(String nom, int valeur)
Description: Supprime une valeur (et attribut si plus de valeur) de la liste des attributs connexions existants.

D.12.4.12 Méthode: toString()
Description: Permet de visualiser une connexion sous forme de chaîne; utilisée pour construire les listes.

D.12.4.13 Méthode: toStringSimple()
Description: Identique à *toString()*, mais de manière moins complète utilisée pour construire les listes.

D.13 Classe *Couple*

D.13.1 Description: Décrit le couple d'une connexion.

D.13.2 Attributs

Nom	Description
Id1	Identificateur du premier géon du couple.
Id2	Identificateur du second géon du couple.

D.13.3 Constructeur

D.13.3.1 Constructeur: Couple(int id1, int id2)
Description: Construit un couple à partir de deux géons.

D.13.4 Méthodes

D.13.4.1 Méthode: equals(Couple couple)
Description: Permet de comparer deux couples.

D.13.4.2 Méthode: getId1()
Description: Récupère l'identificateur du premier géon.

D.13.4.3 Méthode: getId2()
Description: Récupère l'identificateur du second géon.

D.13.4.4 Méthode: toString()
Description: Permet de visualiser un couple sous forme de chaîne; utilisée pour construire les listes.

D.14 Classe *FileManager*

D.14.1 Description: Cette classe s'occupe de la gestion des fichiers d'entrées-sorties.

D.14.2 Attributs

Nom	Description
* Base	Base de travail.
Charger	Charger la base.
Enregistrer	Enregistrer.
Fichier-Attribut-Connexion	Fichier attributs connexions.
Fichier-Attribut-Geon	Fichier attributs géons.
Fichier-Base-Reduite	Fichier base réduite.
Fichier-Geon	Fichier géon.
Fichier-Piece	Fichier pièce.
Fichier-Type	Fichier type.
Fichiers-Tous	Tous les fichiers.
* Repertoire	Répertoire où on enregistre les fichiers de la base de données.

D.14.3 Constructeur

D.14.3.1 Constructeur: FileManager()
Description: Crée le fichier qui s'occupe de la gestion des fichiers d'entrées-sorties.

D.14.4 Méthodes

D.14.4.1 Méthode: LancerFileManager(Base uneBase, int choix, int fichier)
Description: Permet d'enregistrer ou de charger la base à partir de fichiers binaires.

D.14.4.2 Méthode: loadAttributConnexion(String name)
Description: Charge le fichier *AttributConnexion*.

D.14.4.3 Méthode: loadAttributGeon(String name)
Description: Charge le fichier *AttributGeon*.

D.14.4.4 Méthode: loadBaseReduite(String name)
Description: Charge le fichier *BaseReduite*.

D.14.4.5 Méthode: loadPiece(String name)
Description: Charge le fichier *Piece*.

D.14.4.6 Méthode: loadTypeFichier(String name)
Description: Charge le fichier *TypeFichier*.

D.14.4.7 Méthode: loadVueGeon(String name)
Description: Charge le fichier *GeonExistant.*
D.14.4.8 Méthode: SaveAttributConnexion(String name)
Description: Sauvegarde le fichier *AttributConnexion.*
D.14.4.9 Méthode: SaveAttributGeon(String name)
Description: Sauvegarde le fichier *AttributGeon.*
D.14.4.10 Méthode: SaveBaseReduite(String name)
Description: Sauvegarde le fichier BaseReduite.
D.14.4.11 Méthode: SavePiece(String name)
Description: Sauvegarde le fichier *Piece.*
D.14.4.12 Méthode: SaveTypeFichier(String name)
Description: Sauvegarde le fichier *TypeFichier.*
D.14.4.13 Méthode: SaveVueGeon(String name)
Description: Sauvegarde le fichier *GeonExistant.*

D.15 Classe *MyHashtable*

D.15.1 Description: Classe dérivée du tableau d'adressage calculé (*Hashtable)*.

D.15.2 Attributs: Aucun

D.15.3 Constructeur

D.15.3.1 Constructeur: MyHashtable()
Description: Constructeur fait appel au constructeur de *Hashtable*.

D.15.4 Méthodes

D.15.4.1 Méthode: contains[1](Geon geon)
Description: Contains adaptée aux géons.
D.15.4.2 Méthode: containsKey[2](Couple couple)
Description: Containskey adaptée aux couples.
D.15.4.3 Méthode: equals(MyHashtable table)
Description: Permet de comparer deux tableaux d'adressage calculé.

---

[1] Cette fonction est utilisée pour savoir si un géon fait partie du tableau d'adressage calculé (*Hashtable*). Elle retourne la valeur vraie (*True)* si un géon fait partie du tableau et la valeur faux (*False*) s'il n'en fait pas partie.

[2] Cette fonction est utilisée pour savoir si un couple de géons fait partie des clés du tableau d'adressage calculé (*Hashtable*). Elle retourne la valeur vraie (*True)* si le couple fait partie des clés du tableau et la valeur faux (*False*) s'il n'en fait pas partie.

D.16 Classe *Service ou ServiceGest*

D.16.1 Description: Permet de gérer les commandes envoyées sur la douille (*socket*).

D.16.2 Attributs

Nom	Description
Base	Base générale de travail.
BaseReduite	Base réduite de travail.
* ClientPrésent	Vecteur comprenant les numéros de connexions des clients actuellement connectés.
Entree	Canal d'entrée.
EntreeFichier	Entrée de fichier.
Limitation	Vecteur comprenant les numéros des pièces contenues dans la *baseReduite.*
* NbClient	Nombre de clients déjà connectés de 0 à 999.
NumClient	Numéro de connexion du client.
* Recharger	Vecteur comprenant les numéros de connexion des clients qui doivent recharger la *baseReduite.*
Socket	Moyen de communication entre un client et un serveur.
Sortie	Canal de sortie.
SortieFichier	Sortie de fichier.
* Trace	Fichier de trace.

D.16.3 Constructeur

D.16.3.1 Constructeur: Service(Socket socket, Base base, PrintStream trace)
Description: Crée un service, les tampons d'entrée et de sortie et donne un numéro de connexion au client.

D.16.4 Méthode

D.16.4.1 Méthode: run()
Description: Attend et traite la commande envoyée par un client.

D.17 Classe *Serveur*

D.17.1 Description: Permet de gérer les demandes de connexion au serveur de la douille (*socket*).

D.17.2 Attributs

Nom	Description
Base	Base sur laquelle va travailler le serveur.
* FichierTrace	Fichier de trace.
Port	Port de la connexion.
Receptionniste	Serveur de socket.

D.17.3 Constructeur

D.17.3.1 Constructeur: Serveur(int unPort, Base base)
Description: Crée le serveur de socket et lance le fil (T*hread*).

D.17.4 Méthodes

D.17.4.1 Méthode: main(String[])
Description: Charge la base à partir des fichiers s'ils existent et crée le serveur.

D.17.4.2 Méthode: run()
Description: Attend des connexions à la douille et crée un service pour chaque client.

ANNEXE E

LISTE DES REQUÊTES ACCEPTÉES PAR LE SERVEUR

Les requêtes sont envoyées au serveur sous forme de chaîne de caractères comme ci-dessous:

ListeBase
Renvoie la liste des bases réduites sous la forme:
nomBase [liste des étiquettes des pièces faisant partie de la base réduite]

ListePiece
Renvoie la liste des pièces de la base sous la forme:
Piece *numPiece* commentaire *commentaire* \n

ListeFichierGif
Renvoie la liste de tous les fichiers images (Gif) représentant les pièces sous la forme: *numPiece nomFichier* (séparés par des espaces)

ListeGeon *idPiece*
Renvoie la liste des géons d'une pièce sous la forme:
idGeon Geon *etiquetteGeon* \n

ListeConnexion *idPiece*
Renvoie la liste des connexions d'une pièce sous la forme:
Connexion *idGeon1-idGeon2 valeurAttr* \n (à date il n'y a qu'un attribut de connectivité)

ListeFichier *idPiece*
Renvoie la liste des fichiers d'une pièce sous la forme:
nomFichier commentaire date \n

ListeFichier2D *idPiece*
Renvoie la liste des fichiers 2D d'une pièce sous la forme:
nomFichier angleEclairage angleCamera commentaire date \n

ListeAttributGeon *idPiece idGeon*
Renvoie la liste des attributs du géon spécifié sous la forme:
nomAttribut valeurAttribut signification \n

ListeAttributConnexion *idPiece idGeon1-idGeon2*
Renvoie la liste des attributs de la connexion spécifiée sous la forme:
idGeon1-idGeon2 nomAttribut valeurAttribut signification \n

ListeGeonExistant
Renvoie la liste des géons existants sous la forme:
Geon *etiquetteGeon* \n

ListeTypeFichier
Renvoie la liste des types de fichiers existants sous la forme:
nomType * si unique ou *nomType* si non unique

ListeTypeFichier2D
Renvoie la liste des types de fichiers 2D existants sous la forme:
nomType

ListeTypeFichier2DPlus
Renvoie la liste des types de fichiers 2D existants sous la forme:
nomType (Fichier 2D)

ListeTypeFichierNonUnique
Renvoie la liste des types de fichiers non unique existants sous la forme:
nomType

ListeFichierType *nonType*
Renvoie la liste des fichiers du type *nomType* sous la forme:
numPiece nomFichier (séparés par des espaces)

ListeAttributGeonExistant
Renvoie la liste des attributs possibles pour les géons sous la forme:
nomAttribut valeurAttribut signification \n

ListeAttributConnexionExistant
Renvoie la liste des attributs possibles pour les connexions sous la forme:
nomAttribut valeurAttribut signification \n

ContientGeon *etiquetteGeon*
Renvoie la liste des pièces contenant le géon spécifié suivi du nombre d'occurrence dans la pièce sous la forme:
Piece *idPiece* Nb occurrence: *nbOccurence* \n

NbContientGeon *etiquetteGeon*
Renvoie le nombre de pièces contenant le géon spécifié.

MaxGeon
Renvoie la plus grande étiquette géon +1.

AjouterBase *nomBase liste des étiquettes des pièces devant figurer dans la base réduite.*
Ajoute la base réduite dans la liste des bases réduites, renvoie 0 si la base a été ajoutée ou –1 si le nom de la base réduite existe déjà.

SupprimerBase *nomBase*
Supprime la base de la liste des bases réduites, renvoie 0 si la base a été supprimée ou –1 si la base réduite ne fait pas partie de la liste.

AjouterPiece *nomFichierGif nomFichierNeu commentairePiece*
Ajoute la pièce dans la base et renvoie l'étiquette de la pièce créée. N'effectue pas le transfert des fichiers

SupprimerPiece *idPiece*
Supprime la pièce spécifiée, renvoie 0 si OK ou –1 si la pièce n'existe pas.

AjouterGeon *idPiece etiquetteGeon*
Ajoute le géon spécifié à la pièce spécifiée, renvoie l'identificateur du géon ou –1 si la pièce ou le géon n'existe pas.

SupprimerGeon *idPiece idGeon*
Supprime le géon de la pièce spécifiée, renvoie 0 si OK et –1 si la pièce ou le géon n'existe pas

AjouterConnexion *idPiece idGeon1 idGeon2 nomAttr valeurAttr*
Ajoute la connexion à la pièce spécifiée avec comme premier attribut nomAttr de valeur valeurAttr, renvoie 0 si OK et –1 si la pièce n'existe pas ou si la connexion existe déjà (Attention une connexion 1-2 est équivalente à une connexion 2-1).

SupprimerConnexion *idPiece idGeon1 idGeon2*
Supprime la connexion de la pièce spécifiée, renvoie 0 si OK et –1 si la pièce ou la connexion n'existe pas.

AjouterFichier *idPiece nomFichier nomType commentaire*
Ajoute le fichier à la pièce (0 si OK et –1 si la pièce n'existe pas).

SupprimerFichier *idPiece nomFichier*
Supprime le fichier de la pièce spécifiée (0 si OK et –1 si la pièce ou le fichier n'existe pas).

ModifierFichier *idPiece nomFichier commentaire*
Modifie le commentaire et la version du fichier de la pièce spécifiée (0 si OK et –1 si la pièce ou le fichier n'existe pas).

ModifierCommentaireFichier *idPiece nomFichier commentaire*
Modifie le commentaire du fichier, renvoie 0 si OK et –1 si la pièce ou le fichier n'existe pas.

CommentaireFichier *idPiece nomFichier*
Renvoie le commentaire associé au fichier.

AjouterFichier2D *idPiece nomFichier nbAngleEclairage angleEclairage1 angleEclairage2 …angleCamera commentaire*
Ajoute le fichier 2D à la pièce, renvoie 0 si OK et –1 si la pièce n'existe pas.

SupprimerFichier2D *idPiece nomFichier*
Supprime le fichier 2D de la pièce spécifiée, renvoie 0 si OK et –1 si la pièce ou le fichier n'existe pas.

ModifierFichier2D *idPiece nomFichier commentaire*
Modifie le commentaire et la version du fichier 2D de la pièce spécifiée , renvoie 0 si OK et –1 si la pièce ou le fichier n'existe pas.

ModifierCommentaireFichier *idPiece nomFichier commentaire*
Modifie le commentaire du fichier, renvoie 0 si OK et –1 si la pièce ou le fichier n'existe pas.

CommentaireFichier2D *idPiece nomFichier*
Renvoie le commentaire associé au fichier 2D.

ModifierAttributConnexion *idPiece idGeon1 idGeon2 nomAttr valeurAttr*
Ajoute ou modifie la valeur de l'attribut pour la connexion spécifiée, renvoie 0 si OK et –1 si la pièce, la connexion, l'attribut ou la valeur de l'attribut n'existe pas.

SupprimerAttributConnexion *idPiece idGeon1 idGeon2 nomAttr*
Supprime l'attribut de la connexion spécifiée, renvoie 0 si OK et –1 si la pièce, la connexion ou l'attribut n'existe pas.

AjouterType *nomType booleenUnique*
Ajoute un nouveau type ayant comme nom *nomType*, renvoie 0 si OK et –1 s'il existe déjà un type avec ce nom. BooleanUnique est un entier, 0 pour Non unique et 1 pour Unique.

AjouterType2D *nomType*
Ajoute un nouveau type 2D ayant comme nom *nomType*, renvoie 0 si OK et –1 s'il existe déjà un type avec ce nom.

SupprimerType *nomType*
Supprime le type de nom *nomType* de la liste des types existants, renvoie 0 si OK ou –1 si le type n'est pas connu.

SupprimerType2D *nomType*
Supprime le type de nom *nomType* de la liste des types 2D existants, renvoie 0 si OK ou –1 si le type n'est pas connu.

AjouterNouveauGeon *etiquetteGeon*
Ajoute un nouveau géon ayant comme étiquette *etiquetteGeon,* renvoie 0 si OK et –1 s'il existe déjà un géon ayant l'étiquette spécifié ou s'il existe un géon sans attribut.
- Spécifique au gestionnaire.

SupprimerNouveauGeon *etiquetteGeon*
Supprime le géon spécifié de la liste des géons existants, renvoie 0 si OK ou –1 si le géon spécifié n'existe pas.
- Spécifique au gestionnaire.

ModifierAttributGeon *etiquetteGeon nomAttr valeurAttr*
Ajoute ou modifie la valeur de l'attribut pour le géon spécifié, renvoie 0 si OK ou –1 si le géon n'existe pas.
- Spécifique au gestionnaire.

SupprimerAttributGeon *etiquetteGeon nomAttr*
Supprime l'attribut du géon spécifie, renvoie 0 si OK ou –1 si le géon ou l'attribut n'existe pas.
- Spécifique au gestionnaire.

AjouterNouvelAttributGeon *nomAttr valeurAttr commentaire*
Ajoute la nouvelle valeur d'attribut dans la liste des attributs possibles pour les géons, renvoie 0 si OK ou –1 si l'attribut et sa valeur existent déjà.
- Spécifique au gestionnaire.

SupprimerNouvelAttributGeon *nomAttr valeurAttr*
Supprime cette valeur de l'attribut spécifié de la liste des attributs possibles pour les géons, renvoie 0 si OK ou –1 si l'attribut ou sa valeur n'existe pas.
- Spécifique au gestionnaire.

AjouterNouvelAttributConnexion *nomAttr valeurAttr commentaire*
Ajoute la nouvelle valeur d'attribut dans la liste des attributs possibles pour les connexions, renvoie 0 si OK ou –1 si l'attribut et sa valeur existent déjà.
- Spécifique au gestionnaire.

SupprimerNouvelAttributConnexion *nomAttr valeurAttr*
Supprime cette valeur de l'attribut spécifié de la liste des attributs possibles pour les connexions, renvoie 0 si OK ou –1 si l'attribut ou sa valeur n'existe pas.
- Spécifique au gestionnaire.

Remplissage
Crée le fichier remplissage qui contient toutes les commandes pour pouvoir restaurer la base de données.
- Spécifique au gestionnaire.

Telecharger *idPiece nomFichierBase nomFichierDest*
Télécharge le fichier source de la base dans le fichier destination.
Uniquement avec ClientSI ou ClientSIGest[1].

Transferer *nomFichierSource idPiece nomFichierBase*
Transfère le fichier source dans la base de données.
Uniquement avec ClientSI ou ClientSIGest.

Quit
Ferme la douille.

[1] Pour connaître l'utilité des programmes **clientSI** et **clientSIGest ,** voir le détail dans l'annexe C.

ANNEXE F

LES FENÊTRES UTILISÉES POUR LA BASE DE DONNÉES

La conception d'un logiciel ou programme informatique a pour objectif de satisfaire les besoins des utilisateurs. Il faut donc bien comprendre ce qu'il en attend [Hurtubise, 1984] .

Les fenêtres de la base de données sont programmés en langage JAVA en utilisant l'environnement de développement JDK de SUN.

Fenêtre principale

Cette fenêtre contient le menu de la base de données des pièces au complet. Elle est lancée à l'exécution de l'application. Elle offre aux *clients* et *gestionnaires* les choix suivants:

- *Ajout/Modification/Visualisation d'une pièce:*

  Cette partie permet de visualiser la liste des pièces, ajouter une pièce, ajouter des fichiers à des pièces ou accéder à la vue géon.

- *Construction d'une base d'inspection:*

  Cette partie permet de limiter la base de travail ou de créer/supprimer une base réduite.

- *Visualisation des Géons existants:*

  Cette partie permet de visualiser la liste des géons existants et leurs attributs. Ceci permet d'accéder à la liste des pièces contenant le géon.

- *Visualisation des Attributs Géons existants:*

  Cette partie permet de visualiser la liste des attributs géons existants.

- *Visualisation des Attributs Connexion existants:*

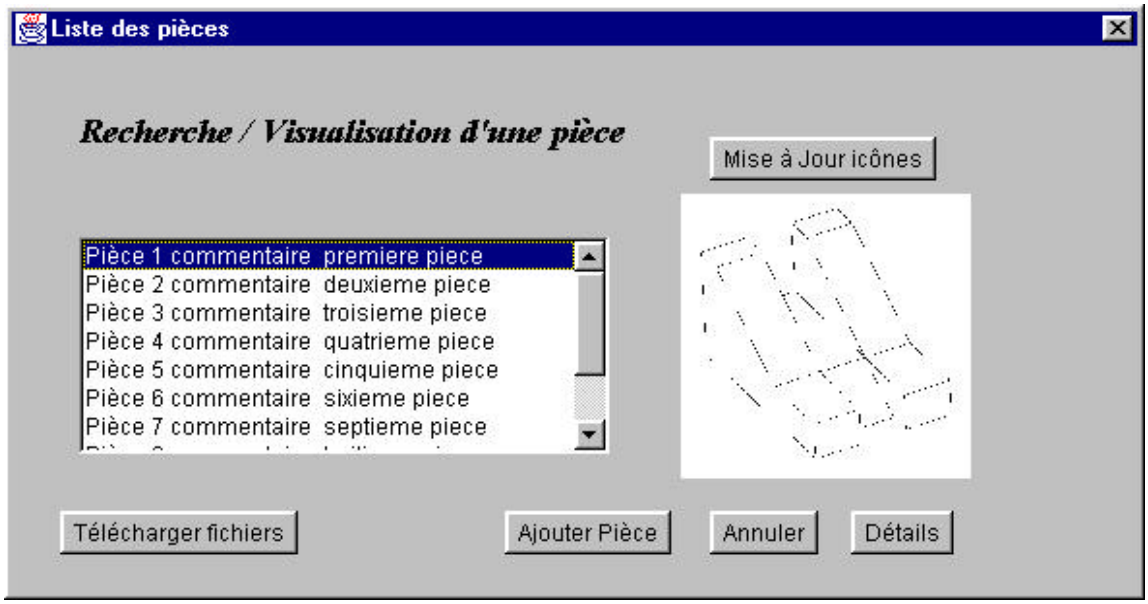
  Cette partie permet de visualiser la liste des attributs connexion existants.

- *Visualisation/Modification des types de fichiers et d'éclairage:*

  Cette partie permet de visualiser, ajouter ou supprimer un type de fichier ou d'éclairage.

F.1 Ajout/Modification/Visualisation d'une pièce

F.1.1 Fenêtre liste des pièces



Cette fenêtre permet de visualiser la liste des pièces de la base de données. Elle contient les boutons suivants:

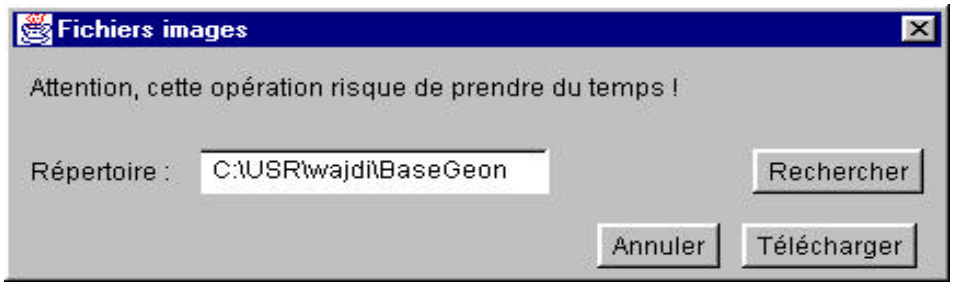
- *Mise à jour icônes:* Permet de télécharger les icônes de toutes les pièces de la base de travail pour pouvoir les afficher dans la fenêtre (ce téléchargement se fait en utilisant le protocole TCP/IP). Cette opération n'est à faire qu'une seule fois par session.

- *Télécharger fichiers:* Permet de télécharger tous les fichiers d'un type à spécifier de toutes les pièces de la base de travail (en utilisant le protocole TCP/IP).

- *Ajouter pièce:* Permet d'ajouter une pièce manufacturée à la base de données.

- *Détails:* Permet de visualiser les informations sur la pièce sélectionnée dans la liste (liste des fichiers, liste des fichiers 2D et vue géon).

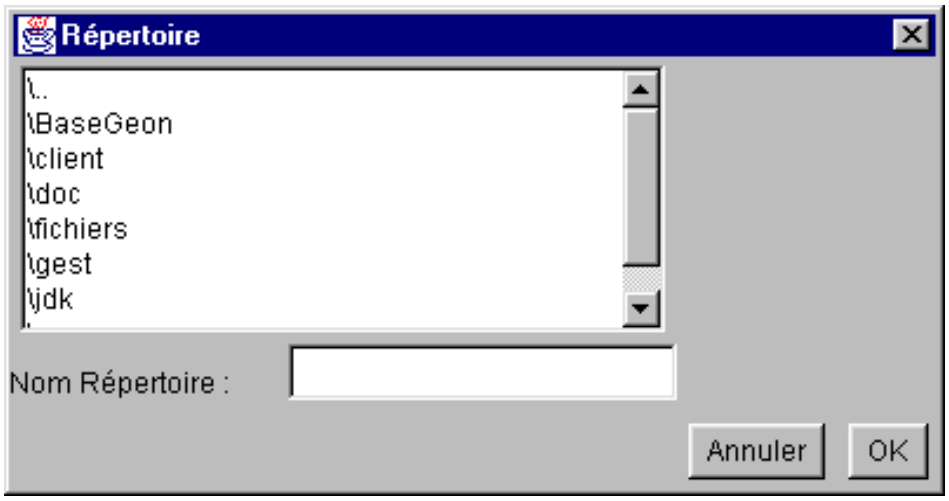
F.1.2 Fenêtre Mise à jour des icônes



Cette fenêtre permet de télécharger les icônes de toutes les pièces de la base de travail. Pour cela, il faut spécifier le répertoire local ou seront téléchargées les icônes des pièces. Elle contient les boutons suivants:

- *Rechercher:* Permet de rechercher le répertoire voulu.
- *Télécharger:* Permet d'effectuer le téléchargement des fichiers.

F.1.3 Bouton Choix du répertoire de destination



Cette fenêtre permet de choisir le répertoire ou seront téléchargées les images des pièces de la base de travail. Pour se promener dans l'arborescence, il suffit de double-cliquer sur le répertoire voulu. Ensuite sélectionner le répertoire et appuyer sur le bouton *OK*.

F.1.4 Fenêtre Télécharger les fichiers

Cette fenêtre permet de télécharger les fichiers du type choisi de toutes les pièces de la base de travail. Pour cela, il faut spécifier le répertoire local ou seront téléchargées les fichiers et le type voulu. Elle contient les boutons suivants:

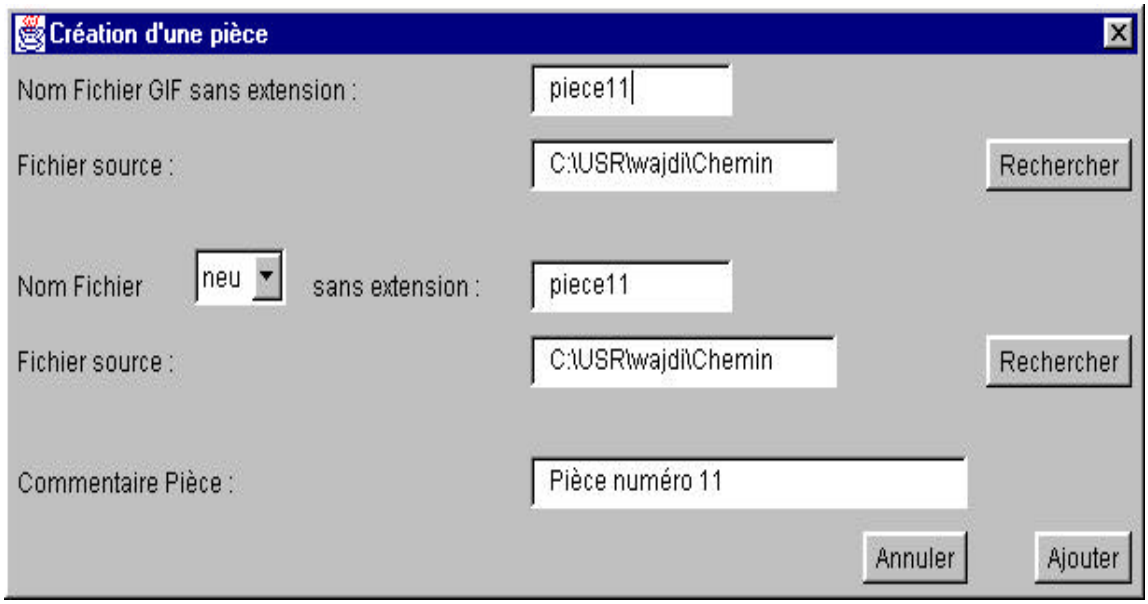
- *Rechercher:* Permet de rechercher le répertoire voulu.

- *Télécharger:* Permet d'effectuer le téléchargement des fichiers.

Remarque: Dans la zone Type de fichier , l'utilisateur peut choisir un type de fichier parmi: *gif *, iges *, im, jpeg, neu *, prt, stl, tiff, ras.*
À noter que les fichiers: *im, jpeg, tiff et ras* sont associés aux fichiers 2D.

Le symbole * suivi des types: *gif, iges* et *neu*, signifie que ces types de fichiers sont obligatoires à chaque nouvelle création d'une pièce. Voir la fenêtre suivante (Ajout d'une pièce).

F.1.5 Fenêtre Ajout d'une pièce



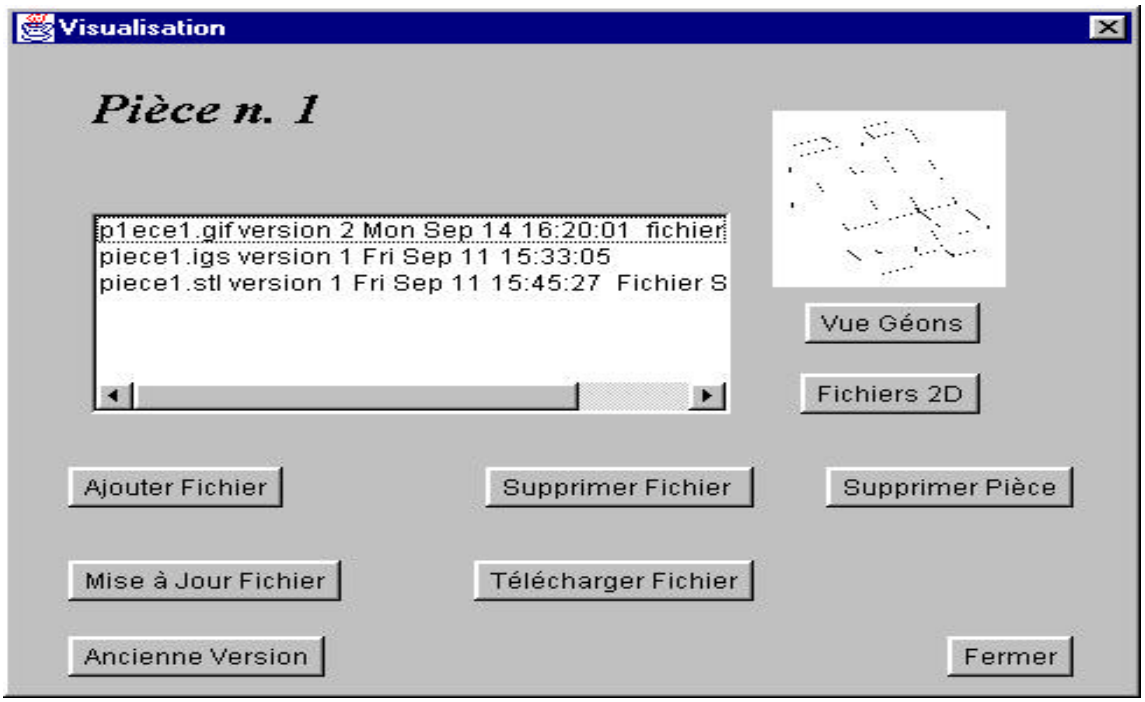
Cette fenêtre permet d'ajouter une pièce à la base de données. Tous les champs de saisie doivent être remplis. Elle contient les boutons suivants:

- *Rechercher:* Les boutons *rechercher* permettent de sélectionner les fichiers sources en local.

- *Ajouter:* Permet d'ajouter la pièce à la base de données.

Remarque: Le menu déroulant du champ Nom Fichier présente deux alternatives: *iges* et *neu.*

F.1.6 Bouton Détails de la pièce

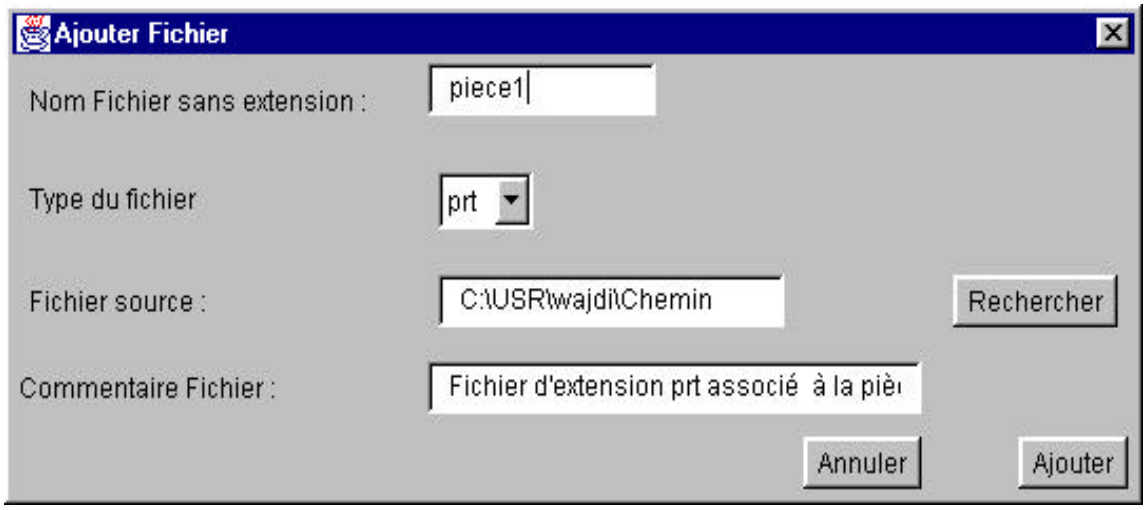


Cette fenêtre permet de visualiser la liste des fichiers de la pièce. Elle contient les boutons suivants:

- *Ajouter Fichier:* Permet d'ajouter un fichier à la pièce.
- *Mise à jour fichier:* Permet de mettre à jour le fichier sélectionné.
- *Supprimer Fichier:* Supprime le fichier sélectionné.
- *Télécharger Fichier:* Permet de télécharger le fichier sélectionné en local.
- *Vue Géons:* Permet de visualiser la vue géon de la pièce.
- *Fichier 2D:* Permet de visualiser la liste des fichiers 2D de la pièce.
- *Supprimer Pièce:* Supprime la pièce de la base de données.
- *Ancienne Version:* Permet de récupérer l'ancienne version du fichier.

Attention: Seul le gestionnaire peut supprimer une pièce de la base de données. La suppression par l'utilisateur, enlève la pièce uniquement de sa base de travail. Pour plus de détails, nous vous référons à l'Annexe B.

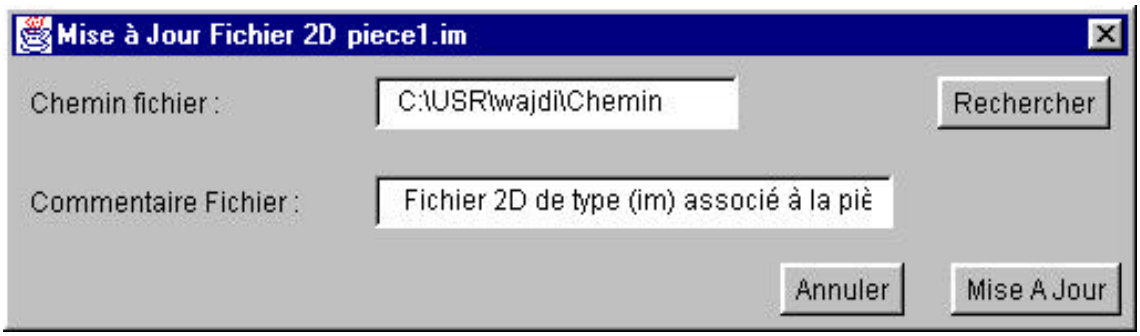
F.1.7 Fenêtre Ajouter un fichier à une pièce



Cette fenêtre permet d'attacher un fichier à une pièce de la base de données. Tous les champs de saisie doivent être remplis. Elle contient les boutons suivants:

- *Rechercher:* Permet de sélectionner le fichier source en local.
- *Ajouter:* Ajoute le fichier à la liste des fichiers de la pièce.

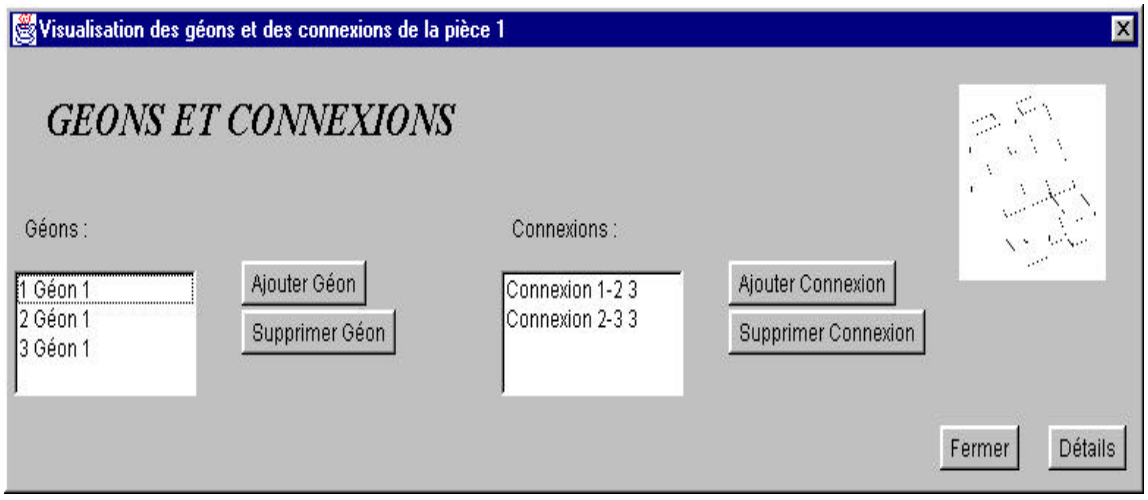
F.1.8 Fenêtre Mise à jour d'un fichier



Cette fenêtre permet de mettre à jour un fichier d'une pièce. Tous les champs de saisie doivent être remplis. Elle contient les boutons suivants:

- *Rechercher:* Permet de sélectionner le fichier source en local.
- *Mise A Jour:* Ce bouton permet de mettre à jour le fichier.

F.1.9 Fenêtre Vue géon



Cette fenêtre permet de visualiser la liste des géons et des connexions d'une pièce. Elle contient les boutons suivants:

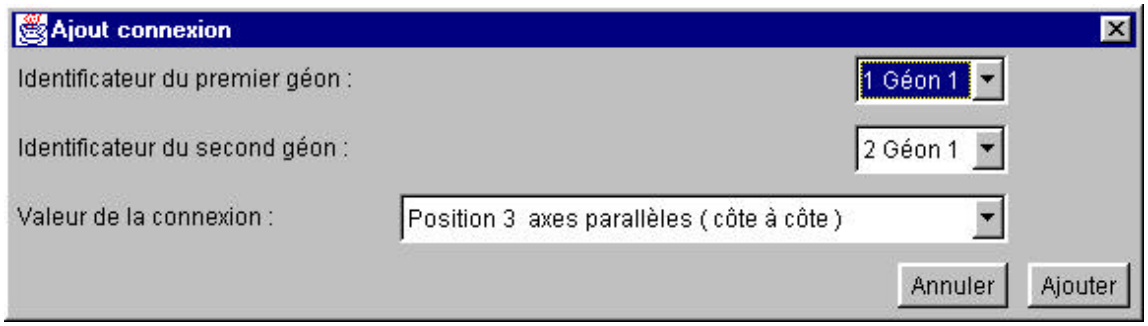
- *Ajouter Géon:* Permet d'ajouter un géon à la pièce.
- *Supprimer Géon:* Supprime le géon sélectionné de la pièce.
- *Ajouter Connexion:* Permet d'ajouter une connexion entre deux géons.
- *Supprimer Connexion:* Supprime la connexion sélectionnée.
- *Détails:* Permet d'avoir des précisions sur le géon ou la connexion sélectionnée.

F.1.9.1 Fenêtre Ajout d'un géon à une pièce

Cette fenêtre permet d'ajouter un géon à une pièce. Le bouton:

- *Ajouter:*
  Ajoute le géon sélectionné à la pièce.

F.1.9.2 Fenêtre Ajout d'une connexion à une pièce



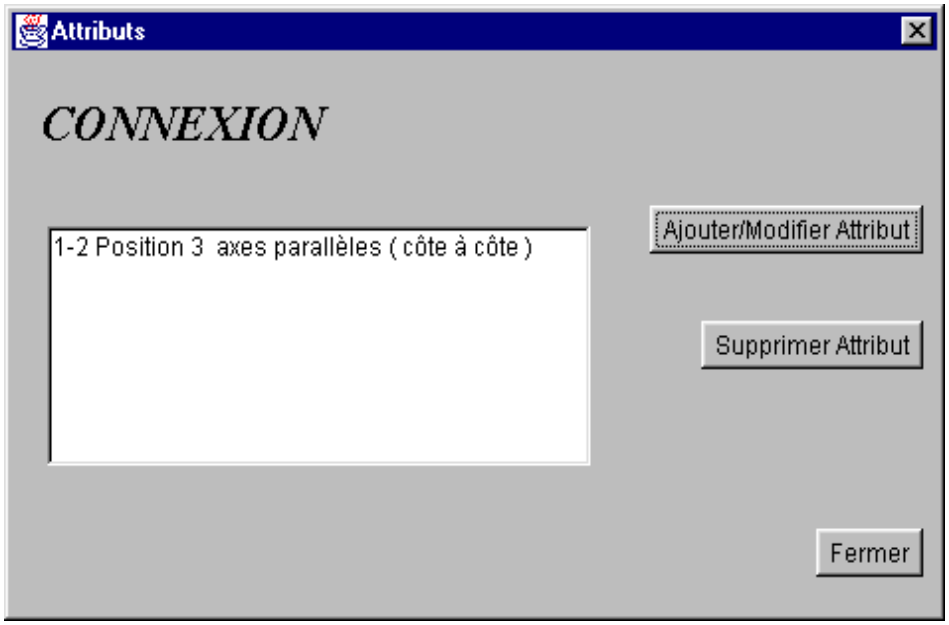
Cette fenêtre permet d'ajouter une connexion entre deux géons à une pièce. Il faut sélectionner deux géons connectés et une valeur de connexion. Le bouton:

- *Ajouter:*
  Permet d'ajouter la connexion entre deux géons dans une pièce.

F.1.10 Fenêtre Liste des pièces contenant un géon

Cette fenêtre permet de visualiser la liste des pièces contenant un géon. Pour connaître l'utilité des boutons, allez voir *les pièces* (¶ F.1.1, page 114).

F.1.11 Fenêtre Attributs d'une connexion

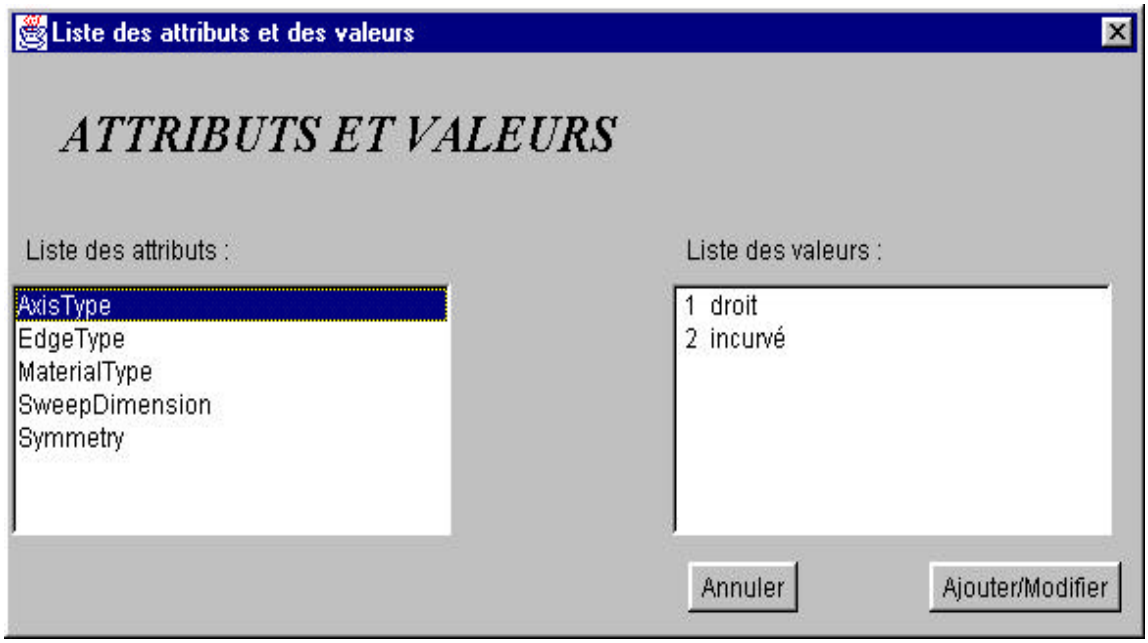


Cette fenêtre permet de visualiser la liste des attributs d'une connexion. Elle contient les boutons suivants:

- *Ajouter Attribut:* Permet d'ajouter ou de modifier un attribut de la connexion.

- *Supprimer Attribut:* Supprime l'attribut sélectionné de la liste des attributs de la connexion.

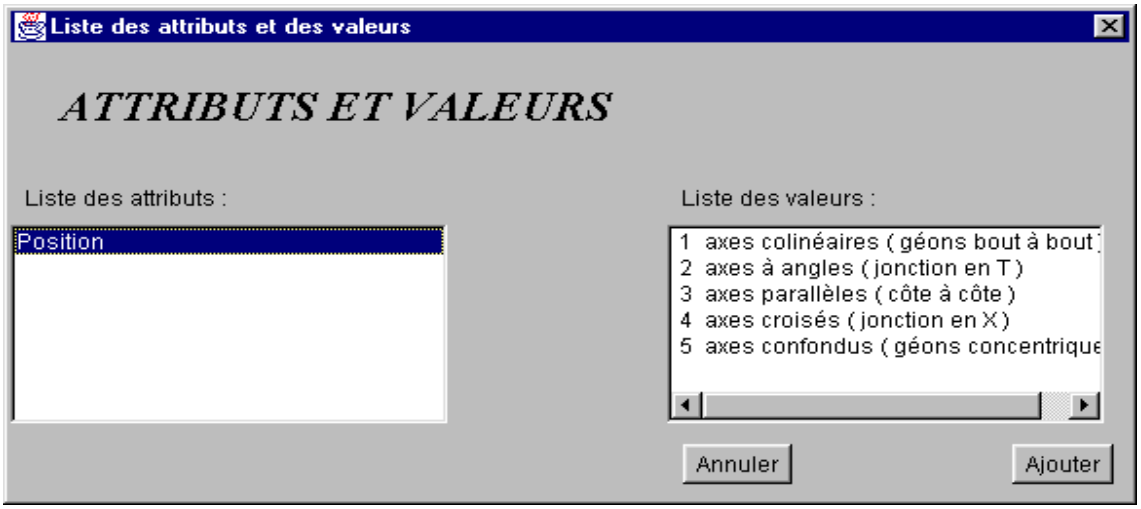
F.1.12 Fenêtre Ajout ou modification d'un attribut de géon/Gestionnaires



Cette fenêtre est spécifique aux *gestionnaires*. Ici, il suffit de sélectionner un attribut dans la liste des attributs et une valeur dans la liste des valeurs. Le bouton:

- *Ajouter*:
  Permet d'ajouter ou de modifier l'attribut au géon.

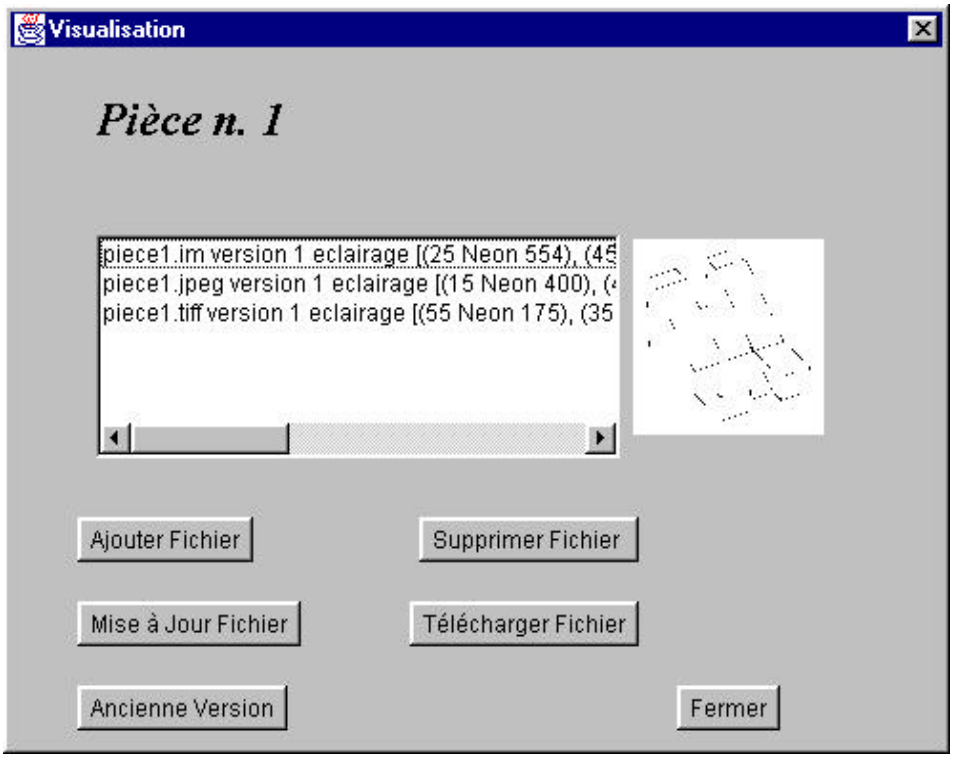
F.1.13 Fenêtre Ajout ou modification d'un attribut de connexion



Dans cette fenêtre, il suffit de sélectionner un attribut dans la liste des attributs et une valeur dans la liste des valeurs. Le bouton:

- *Ajouter:*
  Ajoute ou modifie l'attribut à la connexion.

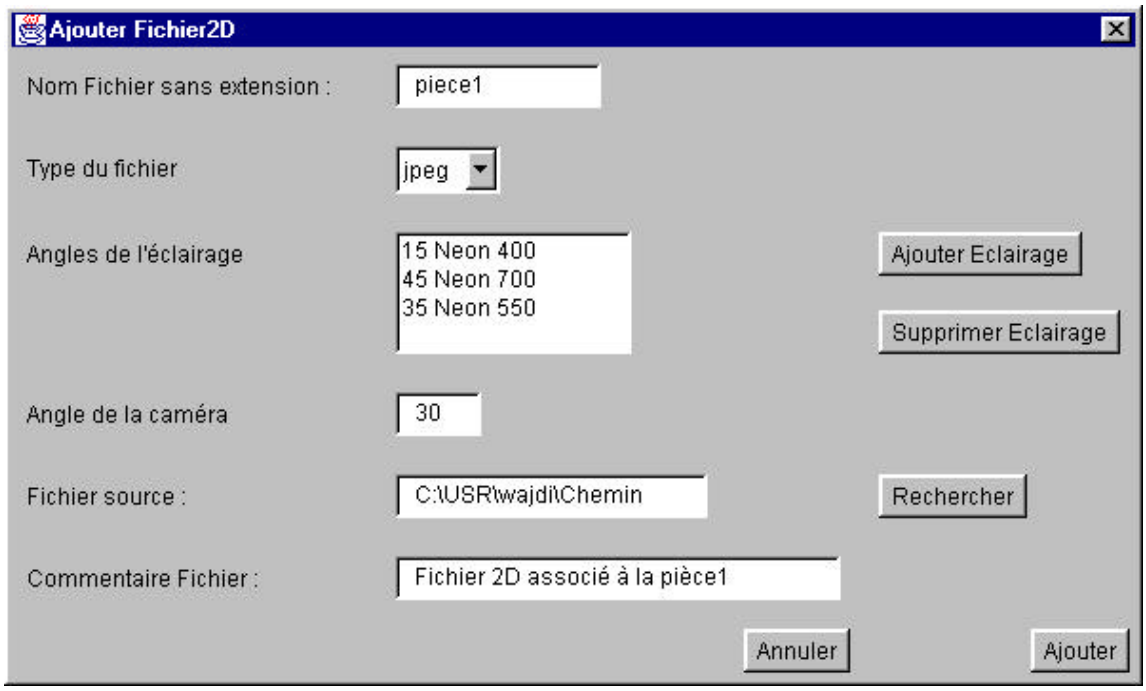
F.1.14 Fenêtre Fichiers 2D



Cette fenêtre permet de visualiser la liste des fichiers dimensionnels (2D) d'une pièce de la base de données. Elle contient les boutons suivants:

- *Ajouter Fichier:* Permet d'ajouter un fichier 2D à la pièce.
- *Mise à jour fichier:* Permet de mettre à jour le fichier 2D sélectionné.
- *Supprimer Fichier:* Supprime le fichier 2D sélectionné.
- *Télécharger Fichier:* Permet de télécharger en local le fichier 2D sélectionné.
- *Ancienne Version:* Permet de récupérer l'ancienne version du fichier.

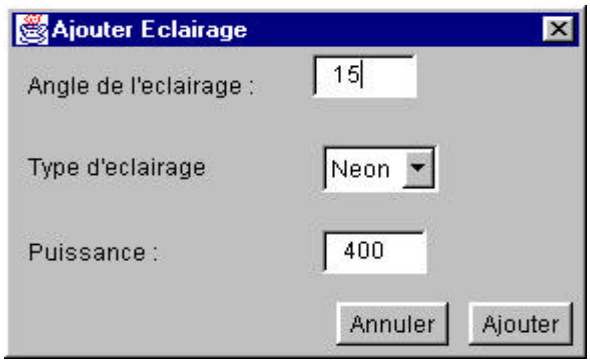
F.1.15 Fenêtre Ajouter un fichier 2D à une pièce



Cette fenêtre permet d'ajouter un fichier 2D à une pièce. Tous les champs de saisie doivent être remplis. Elle contient les boutons suivants:

- *Ajouter Eclairage*: Permet d'ajouter une source d'éclairage dans la liste.
- *Supprimer Eclairage:* Permet de supprimer l'éclairage sélectionné de la liste des éclairages.
- *Rechercher:* Permet de sélectionner le fichier 2D source en local.
- *Ajouter:* Ajoute le fichier 2D à la liste des fichiers 2D de la pièce.

F.1.15.1 Fenêtre Ajouter une source d'éclairage

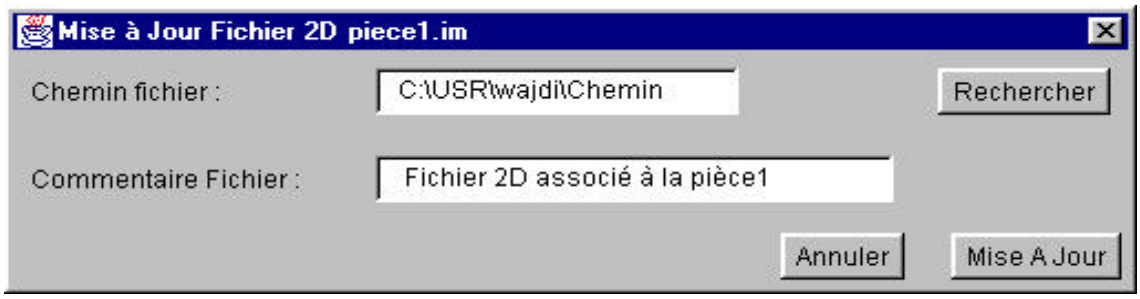


Tous les champs de saisie doivent être remplis. L'angle et la puissance de l'éclairage sont exprimés respectivement en degré et en *Watts*. Le bouton:

- *Ajouter:*
  Permet d'ajouter l'éclairage lors de la création d'un fichier 2D.

Remarque: L'angle de l'éclairage est spécifié par rapport à la verticale.

F.1.15.2 Fenêtre Mise à jour d'un fichier 2D

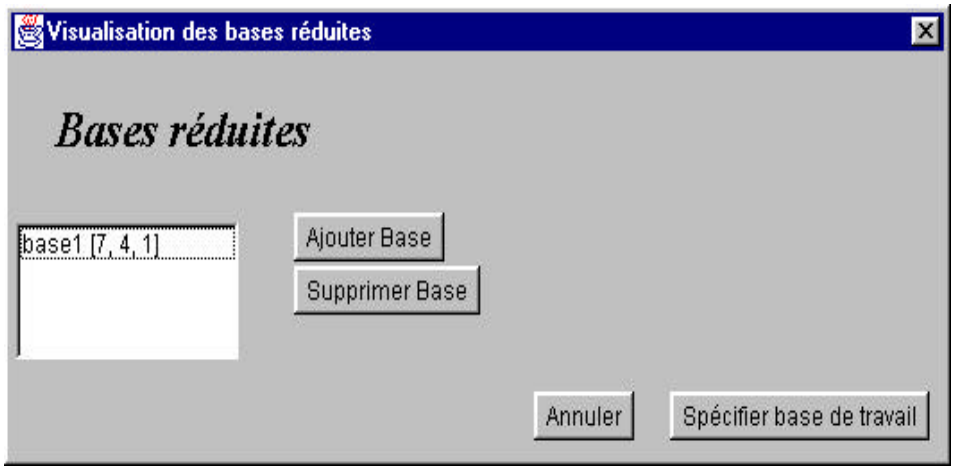


Cette fenêtre permet de mettre à jour un fichier 2D d'une pièce. Elle contient les boutons suivants:

- *Rechercher:* Permet de sélectionner le fichier 2D source en local.
- *Mise A Jour:* Met à jour le fichier 2D.

F.2 Construction d'une base d'inspection

F.2.1 Fenêtre Bases d'inspection

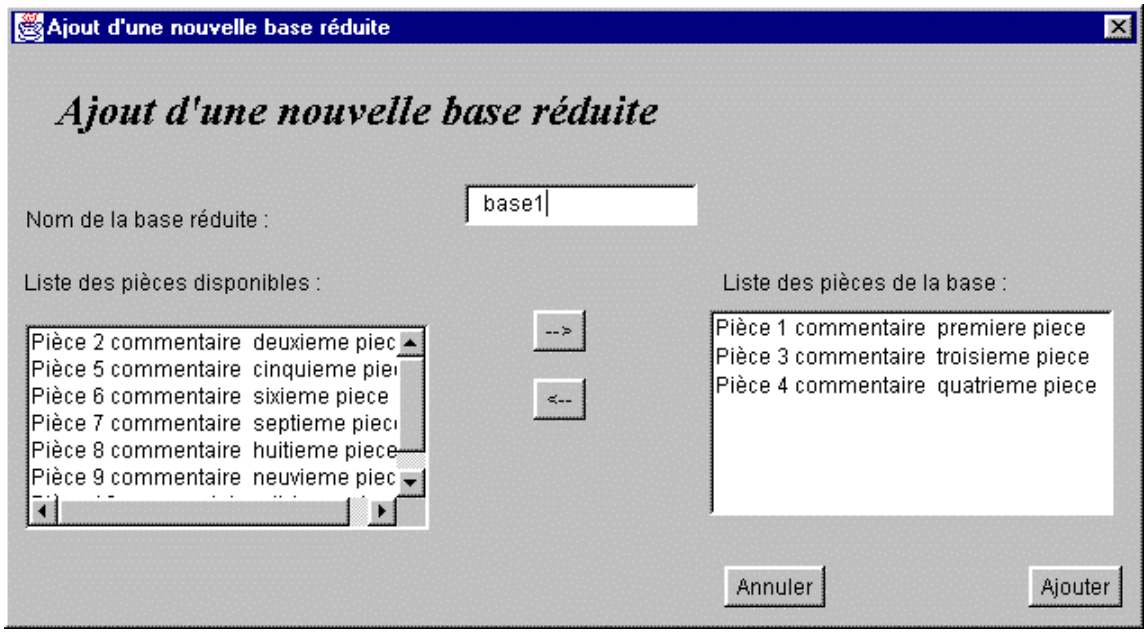


Cette fenêtre permet de visualiser la liste des bases d'inspection[1] existantes. Elle contient les boutons suivants:

- *Ajouter Base:* Permet d'ajouter une nouvelle base réduite à la liste.
- *Supprimer Base:* Supprime la base réduite sélectionnée de la liste.
- *Spécifier base de travail:* Permet de limiter la base de travail à la base réduite sélectionnée.

[1] Une base d'inspection permet de personnaliser la BD en n'incluant pas les pièces susceptibles de se retrouver sur la chaîne d'inspection pour une expérimentation donnée.

F.2.1.1 Fenêtre Ajouter une base réduite

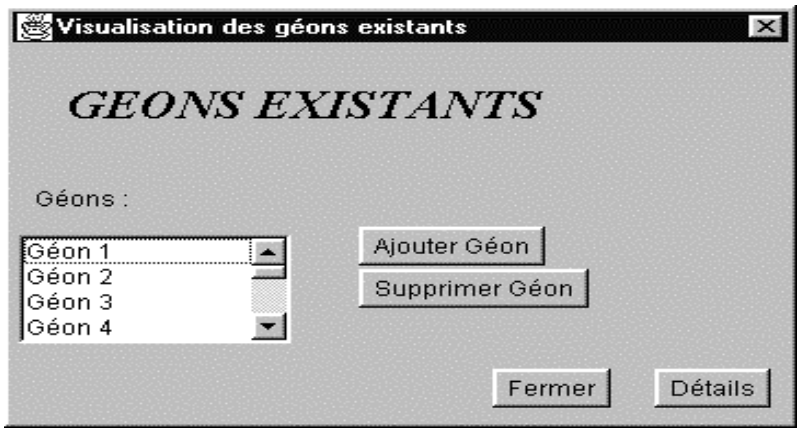


Cette fenêtre permet d'ajouter une base réduite à la liste des bases réduites existantes. Il faut que le nom de la base soit différent des noms des bases réduites existantes. Pour faire passer des pièces dans la liste des pièces de la base réduite, il suffit de sélectionner les pièces et d'appuyer sur le **bouton** --> . Pour supprimer des pièces de la liste de la base réduite, il suffit de sélectionner les pièces voulues et d'appuyer sur le **bouton** <--. Le bouton:

- *Ajouter:*
  Permet d'ajouter la nouvelle base réduite à la liste des bases réduites existantes.

F.3 Visualisation des géons existants

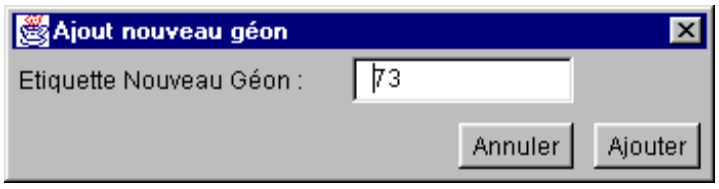
F.3.1 Fenêtre Géons existants/Gestionnaires



Cette fenêtre est spécifique aux *gestionnaires.* Elle permet de visualiser la liste des géons existants. Elle contient les boutons suivants:

- *Ajouter Géon:* Permet d'ajouter un nouveau géon à la liste des géons existants.
- *Supprimer Géon:* Supprime le géon sélectionné de la liste des géons existants.
- *Détails:* Permet de voir la liste des attributs du géon sélectionné (voir ¶ F.3.1.2 à la page suivante).

F.3.1.1 Fenêtre Ajout d'un nouveau géon



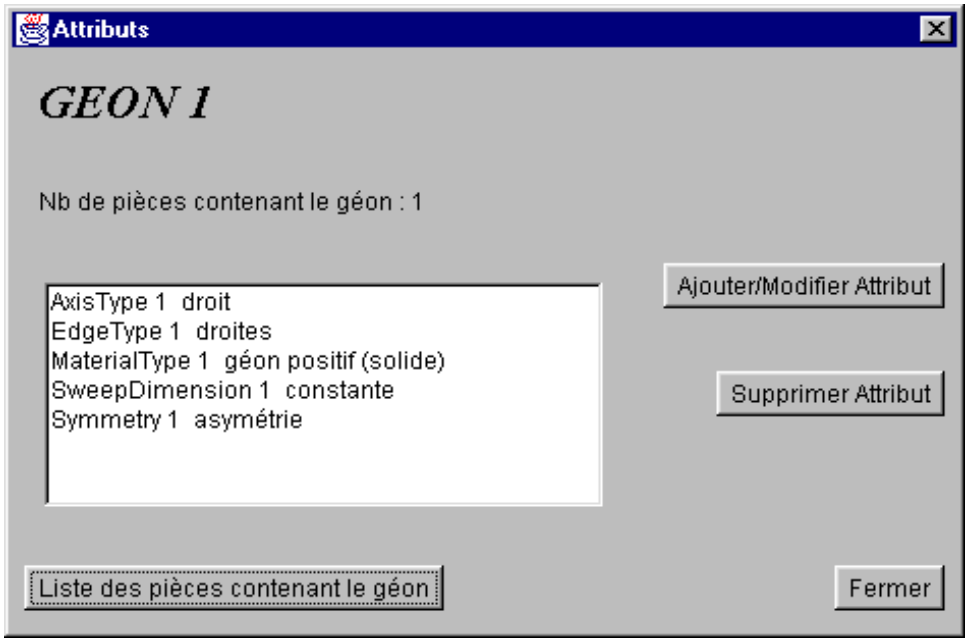
Cette fenêtre est spécifique aux *gestionnaires*, elle permet d'ajouter un nouveau géon à la liste des géons.

- Il suffit de préciser l'étiquette du nouveau géon.
- Il faut que cette étiquette soit différente des étiquettes des géons existants.

Le *bouton:*

- *Ajouter:*
  Permet d'ajouter le géon à la liste des géons existants.

F.3.1.2 Fenêtre Détails sur un géon



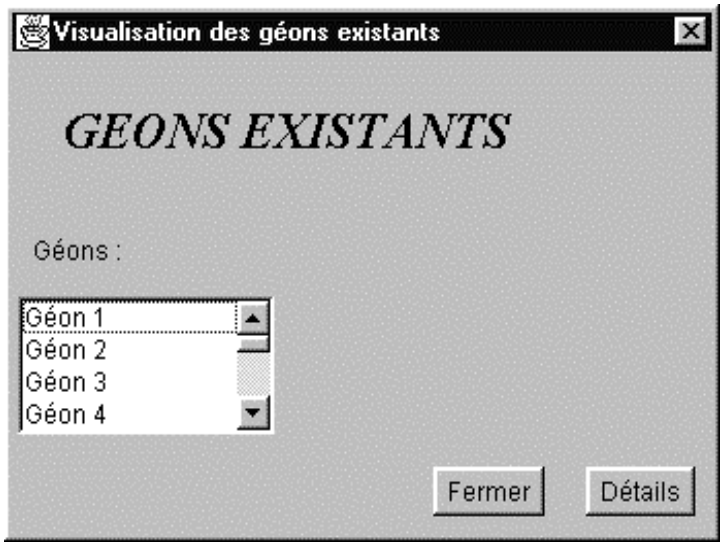
Cette fenêtre est spécifique aux *gestionnaires.* Elle permet de visualiser la liste des attributs d'un géon. Elle contient les boutons suivants:

- *Liste des pièces contenant le géon:* Permet de visualiser la liste des pièces contenant le géon spécifié (voir ¶ F.1.10, page 121).

- *Ajouter/Modifier Attribut:* Permet d'ajouter un attribut à la liste des attributs du géon.

- *Supprimer Attribut:* Supprime l'attribut sélectionné de la liste des attributs du géon.

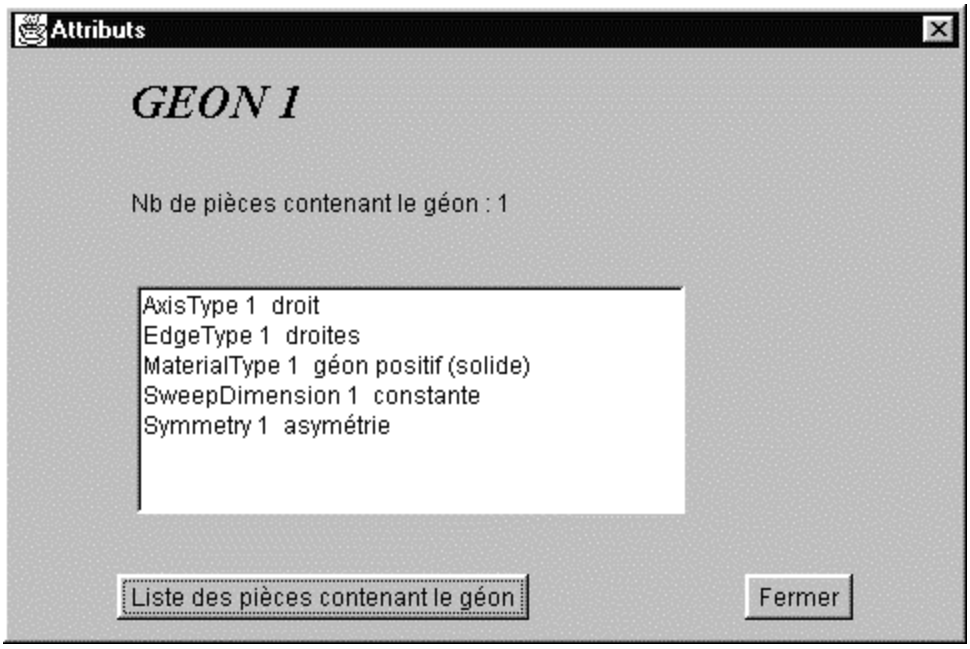
F.3.2 Fenêtre Géons existants/Clients



Cette fenêtre est spécifique aux *clients.* Elle permet de visualiser la liste des géons existants. Le bouton:

- Détails:
  Permet de voir la liste des attributs du géon sélectionné (voir ¶ F.3.2.1 à la page suivante).

F.3.2.1 Fenêtre Liste des attributs d'un géon/Clients

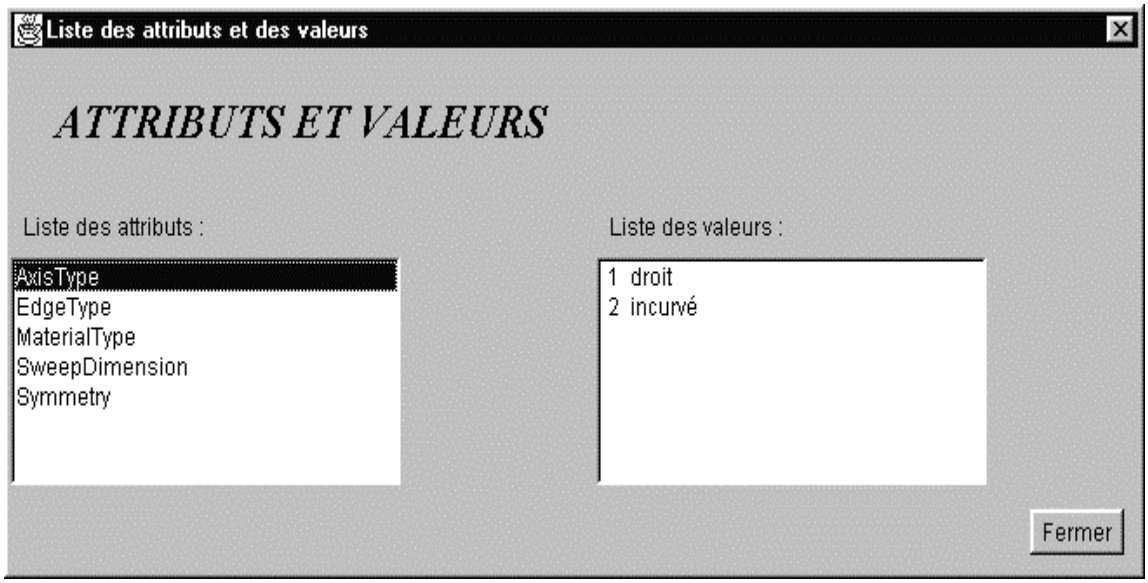


Cette fenêtre est spécifique aux *clients.* Elle permet de visualiser la liste des attributs d'un géon. Le bouton:

- *Liste des pièces contenant le géon:*
  Permet de visualiser la liste des pièces contenant le géon spécifié (voir ¶ F.1.10, Page 121).

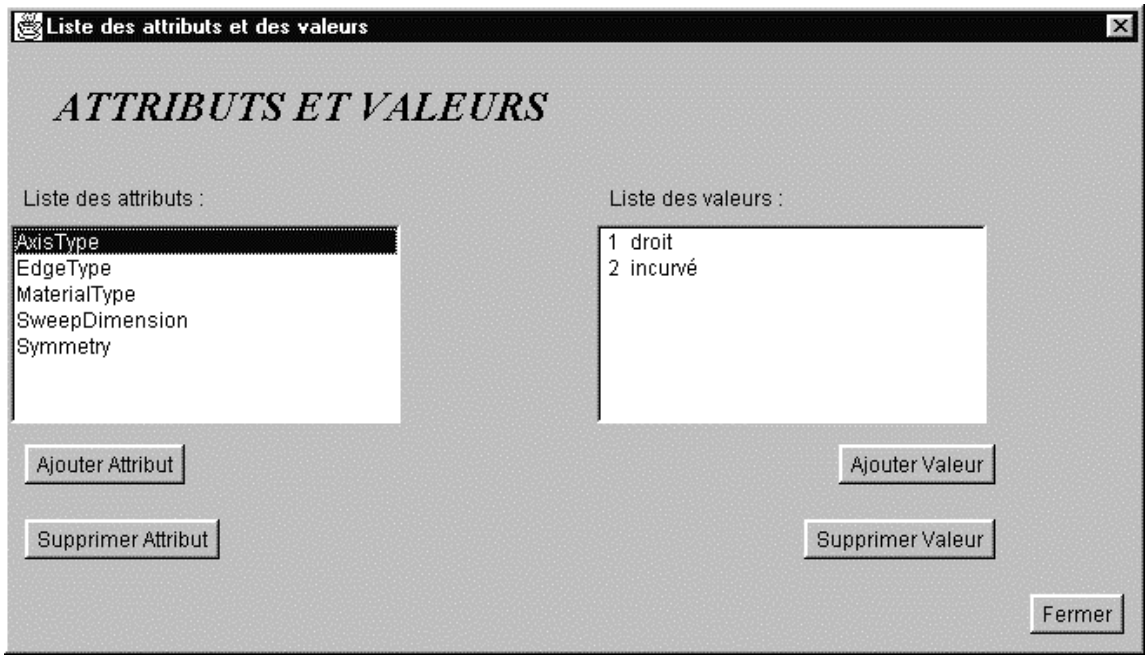
F.4 Visualisation des attributs géon existants

F.4.1 Fenêtre Attributs/Clients



Cette fenêtre est spécifique aux c*lients*. Elle permet de visualiser la liste des attributs géons ou connexions et leurs valeurs.

F.4.2 Fenêtre Ajout ou modification d'un attribut de géon/Gestionnaires



Cette fenêtre est spécifique aux *gestionnaires*. Elle permet de visualiser et modifier la liste des attributs ou connexions et leurs valeurs. Elle contient les boutons suivants:

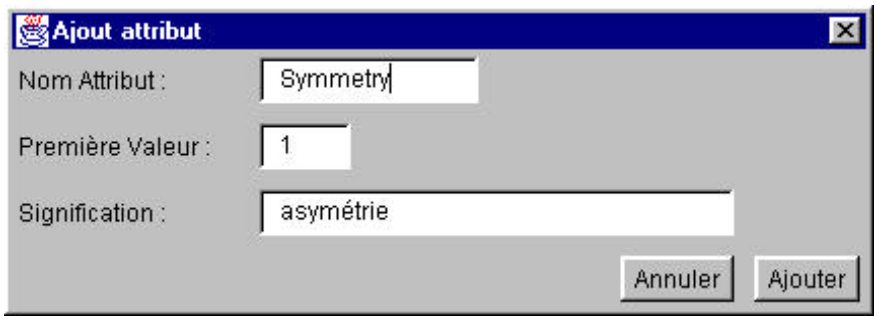
- *Ajouter Attribut:* Permet d'ajouter un attribut à la liste des attributs existants pour le géon ou la connexion.

- *Supprimer Attribut:* Supprime l'attribut sélectionné et toutes ses valeurs de la liste des attributs existants.

- *Ajouter Valeur:* Permet d'ajouter une valeur à l'attribut sélectionné.

- *Supprimer Valeur:* Supprime la valeur sélectionnée des valeurs de l'attribut.

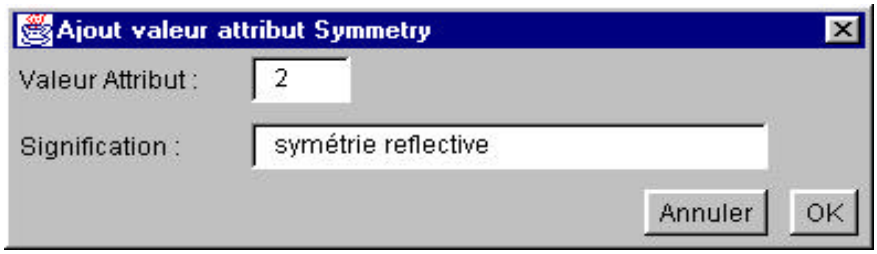
F.4.2.1 Fenêtre Ajout d'un attribut à la liste des attributs existants



Cette fenêtre est spécifique aux *gestionnaires*. Elle permet d'ajouter un attribut à la liste des attributs existants. Il faut spécifier tous les champs de saisie. Le bouton:

- *Ajouter:*
  Ajoute l'attribut à la liste des attributs existants.

F.4.2.2 Fenêtre Ajout d'une valeur à un attribut



Cette fenêtre est spécifique aux *gestionnaires*. Elle permet d'ajouter une valeur à un attribut. Il faut remplir tous les champs de saisie. Le bouton:

- *OK:*
  Ajoute la valeur aux valeurs de l'attribut.

F.5 Visualisation des attributs connexion existant

Cette section hérite des mêmes fenêtres que la section F.4 précédente. Seules, la liste des attributs et la liste des valeurs sont différentes. De plus, elle traite des attributs de connectivité entre 2 géons dans une pièce, tandis que la section précédente traite des attributs des géons existants.

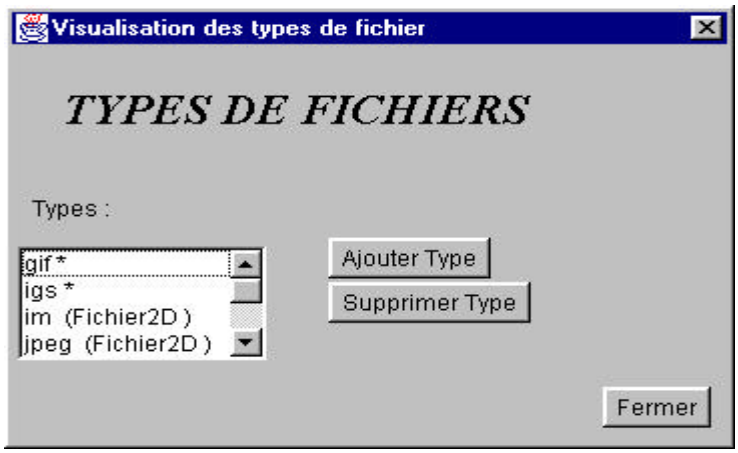
F.6 Visualisation/Modification des types de fichier et d'éclairage

F.6.1 Fenêtre Type de fichier et type d'éclairage

Les boutons:

- *Type de fichier*: Permet de visualiser ou de modifier les types de fichier existants.
- *Type d'éclairage*: Permet de visualiser ou de modifier les types d'éclairage existants.

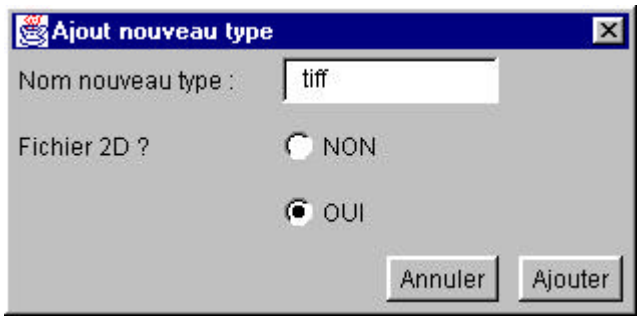
F.6.2 Fenêtre Types de fichiers



Cette fenêtre permet de visualiser la liste des types de fichiers existants. Elle contient les boutons suivants:

- *Ajouter Type:* Permet d'ajouter un nouveau type à cette liste.
- *Supprimer Type:* Supprime le type sélectionné dans la liste.

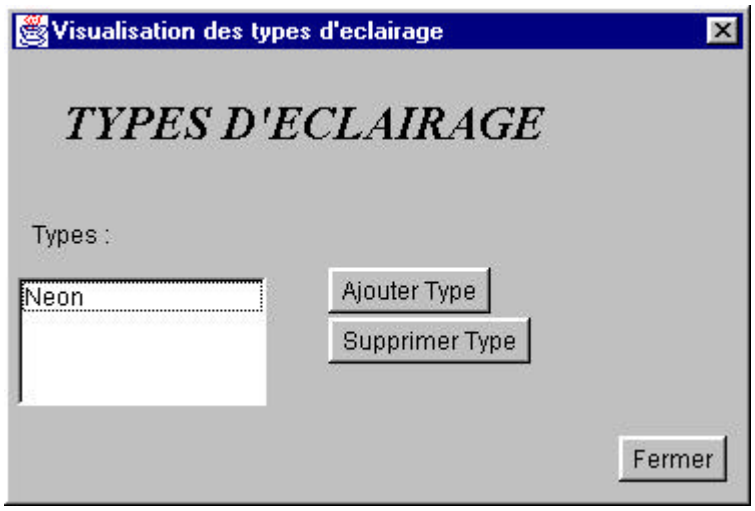
F.6.3 Fenêtre Ajout d'un nouveau type de fichier



Cette fenêtre permet d'ajouter un nouveau type de fichier. Le nom du type doit être différent du nom des types existants. Si le type est Fichier 2D, on pourra avoir plusieurs fichiers de ce type par pièce. Le bouton:

- *Ajouter:*
  Permet d'ajouter un nouveau type de fichier.

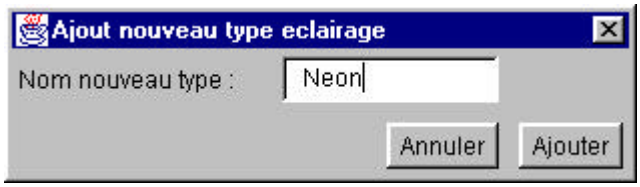
F.6.4 Fenêtre Types d'éclairage



Cette fenêtre permet de visualiser la liste des types d'éclairage existants. Les boutons:

- *Ajouter Type*: Permet d'ajouter un nouveau type d'éclairage à cette liste.
- *Supprimer Type:* Permet de supprimer le type d'éclairage sélectionné dans la liste

F.6.4.1 Fenêtre Ajout d'un nouveau type d'éclairage



Cette fenêtre permet d'ajouter un nouveau type d'éclairage. Le nom du type doit être différent des noms des types d'éclairage existants.

Tout ce que j’arrivai à dire, c’est: «Merci mon dieu». Je devais lui accorder tout le mérite. S’il ne m’avait pas aidé, le travail n’aurait jamais pu être achevé.

«Toute fin est un nouveau commencement!»

Wajdi ABBOUD